LAMMPS Users Manual

Large−scale Atomic/Molecular Massively Parallel Simulator

http://lammps.sandia.gov − Sandia National Laboratories

Copyright (2003) Sandia Corporation. This software and manual is distributed under the GNU General Public License.

Table of Contents

LAMMPS Documentation....1
Version info:....1
1. Introduction....4
1.1 What is LAMMPS....4
1.2 LAMMPS features....5
General features....5
Particle and model types....5
Force fields....5
Atom creation....6
Ensembles, constraints, and boundary conditions....6
Integrators....6
Diagnostics....7
Output....7
Multi−replica models....7
Pre− and post−processing....7
Specialized features....7
1.3 LAMMPS non−features....7
1.4 Open source distribution....9
1.5 Acknowledgments and citations....10
2. Getting Started....15
2.1 What's in the LAMMPS distribution....15
2.2 Making LAMMPS....15
2.3 Making LAMMPS with optional packages....22
2.4 Building LAMMPS as a library....24
2.5 Running LAMMPS....25
2.6 Command−line options....26
2.7 LAMMPS screen output....29
2.8 Tips for users of previous LAMMPS versions....30
3. Commands....31
3.1 LAMMPS input script....31
3.2 Parsing rules....32
3.3 Input script structure....32
3.4 Commands listed by category....34
3.5 Individual commands....34
Fix styles....35
Compute styles....36
Pair_style potentials....36
Bond_style potentials....37
Angle_style potentials....37
Dihedral_style potentials....37
Improper_style potentials....38
Kspace solvers....38
5. Using accelerated CPU and GPU styles....39
5.1 OPT package....40
5.2 USER−OMP package....40
5.3 GPU package....40
5.4 USER−CUDA package....42
5.5 Comparison of GPU and USER−CUDA packages....44

Table of Contents

6. How−to discussions....47
6.1 Restarting a simulation....47
6.2 2d simulations....48
6.3 CHARMM, AMBER, and DREIDING force fields....49
6.4 Running multiple simulations from one input script....50
6.5 Multi−replica simulations....51
6.6 Granular models....52
6.7 TIP3P water model....53
6.8 TIP4P water model....54
6.9 SPC water model....55
6.10 Coupling LAMMPS to other codes....55
6.11 Visualizing LAMMPS snapshots....57
6.12 Triclinic (non−orthogonal) simulation boxes....57
6.13 NEMD simulations....59
6.14 Extended spherical and aspherical particles....60
6.15 Output from LAMMPS (thermo, dumps, computes, fixes, variables)....63
6.16 Thermostatting, barostatting, and computing temperature....67
6.17 Walls....69
6.18 Elastic constants....70
6.19 Library interface to LAMMPS....70
6.20 Calculating thermal conductivity....71
6.21 Calculating viscosity....72
7. Example problems....75
8. Performance &scalability....77
9. Additional tools....78
amber2lmp tool....78
binary2txt tool....79
ch2lmp tool....79
chain tool....79
createatoms tool....79
data2xmovie tool....80
eam database tool....80
eam generate tool....80
eff tool....80
emacs tool....80
ipp tool....80
lmp2arc tool....81
lmp2cfg tool....81
lmp2vmd tool....81
matlab tool....81
micelle2d tool....81
msi2lmp tool....81
pymol_asphere tool....82
python tool....82
reax tool....82
restart2data tool....82
thermo_extract tool....83
vim tool....83

Table of Contents

xmovie tool 83
10. Modifying &extending LAMMPS 84
10.1 Atom styles 85
10.2 Bond, angle, dihedral, improper potentials 86
10.3 Compute styles 87
10.4 Dump styles 87
10.5 Dump custom output options 87
10.6 Fix styles 88
10.7 Input script commands 89
10.8 Kspace computations 89
10.9 Minimization styles 90
10.10 Pairwise potentials 90
10.11 Region styles 90
10.12 Thermodynamic output options 91
10.13 Variable options 91
10.14 Submitting new features for inclusion in LAMMPS 92
11. Python interface to LAMMPS 94
11.1 Extending Python with a serial version of LAMMPS 95
11.2 Creating a shared MPI library 96
11.3 Extending Python with a parallel version of LAMMPS 96
11.4 Extending Python with MPI 97
11.5 Testing the Python−LAMMPS interface 98
11.6 Using LAMMPS from Python 99
11.7 Example Python scripts that use LAMMPS 102
12. Errors 104
12.1 Common problems 104
12.2 Reporting bugs 105
12.3 Error &warning messages 105
Errors: 105
Warnings: 161
13. Future and history 167
13.1 Coming attractions 167
13.2 Past versions 167
angle_style charmm command 170
angle_style class2 command 171
angle_style cg/cmm command 173
angle_coeff command 174
angle_style cosine command 176
angle_style cosine/delta command 177
angle_style cosine/periodic command 178
angle_style cosine/shift command 179
angle_style cosine/shift/exp command 180
angle_style cosine/squared command 181
angle_style harmonic command 182
angle_style hybrid command 183
angle_style none command 185
angle_style command 186
angle_style table command 188

Table of Contents

atom_modify command....190
atom_style command....192
bond_style class2 command....194
bond_coeff command....195
bond_style fene command....197
bond_style fene/expand command....198
bond_style harmonic command....200
bond_style harmonic/shift command....201
bond_style harmonic/shift/cut command....202
bond_style hybrid command....203
bond_style morse command....205
bond_style none command....206
bond_style nonlinear command....207
bond_style quartic command....208
bond_style command....210
bond_style table command....212
boundary command....214
change_box command....215
clear command....216
communicate command....217
compute command....219
compute ackland/atom command....223
compute angle/local command....225
compute atom/molecule command....227
compute bond/local command....229
compute centro/atom command....231
compute cluster/atom command....233
compute cna/atom command....234
compute com command....236
compute com/molecule command....237
compute coord/atom command....239
compute damage/atom command....240
compute dihedral/local command....241
compute displace/atom command....242
compute erotate/asphere command....244
compute erotate/sphere command....245
compute event/displace command....246
compute group/group command....247
compute gyration command....248
compute gyration/molecule command....249
compute heat/flux command....251
compute improper/local command....254
compute ke command....255
compute ke/atom command....256
compute ke/atom/eff command....257
compute ke/eff command....259
compute_modify command....261
compute msd command....262

Table of Contents

compute msd/molecule command....264
compute pair command....266
compute pair/local command....268
compute pe command....270
compute pe/cuda command....270
compute pe/atom command....272
compute pressure command....274
compute pressure/cuda command....274
compute property/atom command....276
compute property/local command....278
compute property/molecule command....280
compute rdf command....281
compute reduce command....283
compute reduce/region command....283
compute slice command....286
compute stress/atom command....288
compute temp command....290
compute temp/cuda command....290
compute temp/asphere command....292
compute temp/com command....295
compute temp/deform command....297
compute temp/deform/eff command....299
compute temp/eff command....300
compute temp/partial command....302
compute temp/partial/cuda command....302
compute temp/profile command....304
compute temp/ramp command....306
compute temp/region command....308
compute temp/region/eff command....310
compute temp/rotate command....311
compute temp/sphere command....313
compute ti command....315
create_atoms command....317
create_box command....320
delete_atoms command....322
delete_bonds command....324
dielectric command....326
dihedral_style charmm command....327
dihedral_style class2 command....329
dihedral_coeff command....332
dihedral_style cosine/shift/exp command....334
dihedral_style harmonic command....335
dihedral_style helix command....336
dihedral_style hybrid command....337
dihedral_style multi/harmonic command....339
dihedral_style none command....340
dihedral_style opls command....341
dihedral_style command....342

Table of Contents

dimension command....344
displace_atoms command....345
displace_box command....347
dump command....350
dump image command....350
dump image command....357
dump_modify command....364
echo command....373
fix command....374
fix adapt command....378
fix addforce command....382
fix addforce/cuda command....382
fix addtorque command....385
fix atc command....387
fix ave/atom command....391
fix ave/correlate command....393
fix ave/histo command....398
fix ave/spatial command....403
fix ave/time command....408
fix aveforce command....413
fix aveforce/cuda command....413
fix bond/break command....415
fix bond/create command....418
fix bond/swap command....421
fix box/relax command....424
fix deform command....429
fix deposit command....435
fix drag command....438
fix dt/reset command....439
fix efield command....441
fix enforce2d command....442
fix enforce2d/cuda command....442
fix evaporate command....443
fix external command....445
fix freeze command....447
fix freeze/cuda command....447
fix gcmc command....449
fix gravity command....452
fix gravity/cuda command....452
fix heat command....454
fix imd command....456
fix indent command....459
fix langevin command....462
fix langevin/eff command....466
fix lineforce command....468
fix_modify command....469
fix momentum command....470
fix move command....472

Table of Contents

fix msst command 475
fix neb command 478
fix nvt command 480
fix nvt/cuda command 480
fix npt command 480
fix npt/cuda command 480
fix nph command 480
fix nvt/eff command 488
fix npt/eff command 488
fix nph/eff command 488
fix nph/asphere command 491
fix nph/sphere command 493
fix npt/asphere command 495
fix npt/sphere command 498
fix nve command 500
fix nve/cuda command 500
fix nve/asphere command 501
fix nve/eff command 502
fix nve/limit command 503
fix nve/noforce command 505
fix nve/sphere command 506
fix nvt/asphere command 508
fix nvt/sllod command 510
fix nvt/sllod/eff command 512
fix nvt/sphere command 514
fix orient/fcc command 516
fix planeforce command 520
fix poems 521
fix pour command 523
fix press/berendsen command 525
fix print command 528
fix qeq/comb command 530
fix qeq/reax command 532
fix reax/bonds command 534
fix recenter command 535
fix restrain command 537
fix rigid command 539
fix rigid/nve command 539
fix rigid/nvt command 539
fix setforce command 545
fix setforce/cuda command 545
fix shake command 547
fix shake/cuda command 547
fix smd command 549
fix spring command 552
fix spring/rg command 554
fix spring/self command 556
fix srd command 557

Table of Contents

fix store/force command....562
fix store/state command....563
fix temp/berendsen command....565
fix temp/berendsen/cuda command....565
fix temp/rescale command....568
fix temp/rescale/cuda command....568
fix temp/rescale/limit/cuda command....568
fix temp/rescale/eff command....571
fix thermal/conductivity command....573
fix tmd command....576
fix ttm command....578
fix viscosity command....581
fix viscous command....584
fix viscous/cuda command....584
fix wall/lj93 command....586
fix wall/lj126 command....586
fix wall/colloid command....586
fix wall/harmonic command....586
fix wall/gran command....590
fix wall/reflect command....593
fix wall/region command....596
fix wall/srd command....599
group command....602
if command....604
improper_style class2 command....607
improper_coeff command....609
improper_style cvff command....611
improper_style harmonic command....612
improper_style hybrid command....613
improper_style none command....614
improper_style command....615
improper_style umbrella command....617
include command....619
jump command....620
kspace_modify command....622
kspace_style command....624
label command....627
lattice command....628
log command....631
mass command....632
min_modify command....634
min_style command....635
minimize command....637
neb command....641
neigh_modify command....646
neighbor command....649
newton command....651
next command....652

Table of Contents

orient command....654
origin command....655
package command....656
pair_style adp command....658
pair_style airebo command....661
pair_style rebo command....661
pair_style awpmd/cut command....664
pair_style born command....666
pair_style born/coul/long command....666
pair_style born/coul/long/cuda command....666
pair_style buck command....669
pair_style buck/cuda command....669
pair_style buck/coul/cut command....669
pair_style buck/coul/cut/cuda command....669
pair_style buck/coul/long command....669
pair_style buck/coul/long/cuda command....669
pair_style buck/coul command....672
pair_style lj/charmm/coul/charmm command....675
pair_style lj/charmm/coul/charmm/cuda command....675
pair_style lj/charmm/coul/charmm/implicit command....675
pair_style lj/charmm/coul/charmm/implicit/cuda command....675
pair_style lj/charmm/coul/long command....675
pair_style lj/charmm/coul/long/cuda command....675
pair_style lj/charmm/coul/long/gpu command....675
pair_style lj/charmm/coul/long/opt command....675
pair_style lj/class2 command....679
pair_style lj/class2/cuda command....679
pair_style lj/class2/gpu command....679
pair_style lj/class2/coul/cut command....679
pair_style lj/class2/coul/cut/cuda command....679
pair_style lj/class2/coul/long command....679
pair_style lj/class2/coul/long/cuda command....679
pair_style lj/class2/coul/long/gpu command....679
pair_style cg/cmm command....682
pair_style cg/cmm/cuda command....682
pair_style cg/cmm/gpu command....682
pair_style cg/cmm/coul/cut command....682
pair_style cg/cmm/coul/cut/cuda command....682
pair_style cg/cmm/coul/debye/cuda command....682
pair_style cg/cmm/coul/long command....682
pair_style cg/cmm/coul/long/gpu command....682
pair_style cg/cmm/coul/long/cuda command....682
pair_coeff command....686
pair_style colloid command....689
pair_style comb command....692
pair_style coul/cut command....696
pair_style coul/debye command....696
pair_style coul/long command....696

Table of Contents

pair_style coul/long/gpu command....696
pair_style dipole/cut command....699
pair_style dipole/sf command....699
pair_style dpd command....704
pair_style dpd/tstat command....704
pair_style dsmc command....707
pair_style eam command....709
pair_style eam/cuda command....709
pair_style eam/opt command....709
pair_style eam/alloy command....709
pair_style eam/alloy/cuda command....709
pair_style eam/alloy/opt command....709
pair_style eam/cd command....709
pair_style eam/fs command....709
pair_style eam/fs/cuda command....709
pair_style eam/fs/opt command....709
pair_style eff/cut command....716
pair_style eim command....720
pair_style gauss command....723
pair_style gayberne command....725
pair_style gayberne/gpu command....725
pair_style gran/hooke command....729
pair_style gran/cuda command....729
pair_style gran/hooke/history command....729
pair_style gran/hertz/history command....729
pair_style lj/gromacs command....733
pair_style lj/gromacs/cuda command....733
pair_style lj/gromacs/coul/gromacs command....733
pair_style lj/gromacs/coul/gromacs/cuda command....733
pair_style hbond/dreiding/lj command....736
pair_style hbond/dreiding/morse command....736
pair_style hybrid command....740
pair_style hybrid/overlay command....740
pair_style lj/cut command....744
pair_style lj/cut/cuda command....744
pair_style lj/cut/experimental/cuda command....744
pair_style lj/cut/gpu command....744
pair_style lj/cut/opt command....744
pair_style lj/cut/coul/cut command....744
pair_style lj/cut/coul/cut/cuda command....744
pair_style lj/cut/coul/cut/gpu command....744
pair_style lj/cut/coul/debye command....744
pair_style lj/cut/coul/debye/cuda command....744
pair_style lj/cut/coul/long command....744
pair_style lj/cut/coul/long/cuda command....744
pair_style lj/cut/coul/long/gpu command....744
pair_style lj/cut/coul/long/tip4p command....744
pair_style lj96/cut command....748

Table of Contents

pair_style lj96/cut/cuda command....748
pair_style lj96/cut/gpu command....748
pair_style lj/coul command....750
pair_style lj/expand command....753
pair_style lj/expand/cuda command....753
pair_style lj/expand/gpu command....753
pair_style lj/sf command....755
pair_style lj/smooth command....757
pair_style lj/smooth/cuda command....757
pair_style lubricate command....759
pair_style meam command....762
pair_modify command....768
pair_style morse command....771
pair_style morse/cuda command....771
pair_style morse/gpu command....771
pair_style morse/opt command....771
pair_style none command....773
pair_style peri/pmb command....774
pair_style peri/lps command....774
pair_style reax command....776
pair_style reax/c command....779
pair_style resquared command....783
pair_style resquared/gpu command....783
pair_style soft command....787
pair_style command....789
pair_style sw command....792
pair_style table command....795
pair_style tersoff command....798
pair_style tersoff/zbl command....802
pair_write command....807
pair_style yukawa command....809
pair_style yukawa/colloid command....811
prd command....813
print command....817
processors command....818
read_data command....819
read_restart command....829
region command....831
replicate command....835
reset_timestep command....836
restart command....837
run command....839
run_style command....842
set command....845
shell command....849
special_bonds command....851
suffix command....854
tad command....855

Table of Contents

temper command 859
thermo command 861
thermo_modify command 862
thermo_style command 865
timestep command 870
uncompute command 871
undump command 872
unfix command 873
units command 874
variable command 877
- Math Operators 881
- Math Functions 881
- Group and Region Functions 883
- Special Functions 883
- Atom Values and Vectors 884
- Compute References 884
- Fix References 884
- Variable References 885

velocity command 889
write_restart command 892

LAMMPS Documentation

Version info:

The LAMMPS "version" is the date when it was released, such as 1 May 2010. LAMMPS is updated continuously. Whenever we fix a bug or add a feature, we release it immediately, and post a notice on this page of the WWW site. Each dated copy of LAMMPS contains all the features and bug−fixes up to and including that version date. The version date is printed to the screen and logfile every time you run LAMMPS. It is also in the file src/version.h and in the LAMMPS directory name created when you unpack a tarball.

- If you browse the HTML doc pages on the LAMMPS WWW site, they always describe the most current version of LAMMPS.
- If you browse the HTML doc pages included in your tarball, they describe the version you have.
- The PDF file on the WWW site or in the tarball is updated about once per month. This is because it is large, and we don't want it to be part of very patch.
- There is also a Developer.pdf file in the doc directory, which describes the internal structure and algorithms of LAMMPS.

LAMMPS stands for Large−scale Atomic/Molecular Massively Parallel Simulator.

LAMMPS is a classical molecular dynamics simulation code designed to run efficiently on parallel computers. It was developed at Sandia National Laboratories, a US Department of Energy facility, with funding from the DOE. It is an open−source code, distributed freely under the terms of the GNU Public License (GPL).

The primary developers of LAMMPS are Steve Plimpton, Aidan Thompson, and Paul Crozier who can be contacted at sjplimp,athomps,pscrozi at sandia.gov. The LAMMPS WWW Site at http://lammps.sandia.gov has more information about the code and its uses.

The LAMMPS documentation is organized into the following sections. If you find errors or omissions in this manual or have suggestions for useful information to add, please send an email to the developers so we can improve the LAMMPS documentation.

Once you are familiar with LAMMPS, you may want to bookmark this page at Section_commands.html#comm since it gives quick access to documentation for all LAMMPS commands.

PDF file of the entire manual, generated by htmldoc

1. Introduction
   1.1 What is LAMMPS
   1.2 LAMMPS features
   1.3 LAMMPS non−features
   1.4 Open source distribution
   1.5 Acknowledgments and citations
2. Getting started
   2.1 What's in the LAMMPS distribution
   2.2 Making LAMMPS
   2.3 Making LAMMPS with optional packages
   2.4 Building LAMMPS as a library
   2.5 Running LAMMPS
   2.6 Command−line options
   2.7 Screen output
   2.8 Tips for users of previous versions

3. Commands
   3.1 LAMMPS input script
   3.2 Parsing rules
   3.3 Input script structure
   3.4 Commands listed by category
   3.5 Commands listed alphabetically
4. Packages
   4.1 Standard packages
   4.2 User packages
5. Using accelerated CPU and GPU styles
   5.1 OPT package
   5.2 USER−OMP package
   5.3 GPU package
   5.4 USER−CUDA package
   5.5 Comparison of GPU and USER−CUDA packages
6. How−to discussions
   6.1 Restarting a simulation
   6.2 2d simulations
   6.3 CHARMM and AMBER force fields
   6.4 Running multiple simulations from one input script
   6.5 Multi−replica simulations
   6.6 Granular models
   6.7 TIP3P water model
   6.8 TIP4P water model
   6.9 SPC water model
   6.10 Coupling LAMMPS to other codes
   6.11 Visualizing LAMMPS snapshots
   6.12 Triclinic (non−orthogonal) simulation boxes
   6.13 NEMD simulations
   6.14 Extended spherical and aspherical particles
   6.15 Output from LAMMPS (thermo, dumps, computes, fixes, variables)
   6.16 Thermostatting, barostatting, and compute temperature
   6.17 Walls
   6.18 Elastic constants
   6.19 Library interface to LAMMPS
   6.20 Calculating thermal conductivity
   6.21 Calculating viscosity
7. Example problems
8. Performance &scalability
9. Additional tools
10. Modifying &extending LAMMPS
   10.1 Atom styles
   10.2 Bond, angle, dihedral, improper potentials
   10.3 Compute styles
   10.4 Dump styles
   10.5 Dump custom output options
   10.6 Fix styles
   10.7 Input script commands
   10.8 Kspace computations
   10.9 Minimization styles
   10.10 Pairwise potentials
   10.11 Region styles

10.12 Thermodynamic output options
10.13 Variable options
10.14 Submitting new features for inclusion in LAMMPS
11. Python interface
11.1 Extending Python with a serial version of LAMMPS
11.2 Creating a shared MPI library
11.3 Extending Python with a parallel version of LAMMPS
11.4 Extending Python with MPI
11.5 Testing the Python−LAMMPS interface
11.6 Using LAMMPS from Python
11.7 Example Python scripts that use LAMMPS
12. Errors
12.1 Common problems
12.2 Reporting bugs
12.3 Error &warning messages
13. Future and history
13.1 Coming attractions
13.2 Past versions

1. Introduction

These sections provide an overview of what LAMMPS can and can't do, describe what it means for LAMMPS to be an open−source code, and acknowledge the funding and people who have contributed to LAMMPS over the years.

1.1 What is LAMMPS
1.2 LAMMPS features
1.3 LAMMPS non−features
1.4 Open source distribution
1.5 Acknowledgments and citations

1.1 What is LAMMPS

LAMMPS is a classical molecular dynamics code that models an ensemble of particles in a liquid, solid, or gaseous state. It can model atomic, polymeric, biological, metallic, granular, and coarse−grained systems using a variety of force fields and boundary conditions.

For examples of LAMMPS simulations, see the Publications page of the LAMMPS WWW Site.

LAMMPS runs efficiently on single−processor desktop or laptop machines, but is designed for parallel computers. It will run on any parallel machine that compiles C++ and supports the MPI message−passing library. This includes distributed− or shared−memory parallel machines and Beowulf−style clusters.

LAMMPS can model systems with only a few particles up to millions or billions. See this section for information on LAMMPS performance and scalability, or the Benchmarks section of the LAMMPS WWW Site.

LAMMPS is a freely−available open−source code, distributed under the terms of the GNU Public License, which means you can use or modify the code however you wish. See this section for a brief discussion of the open−source philosophy.

LAMMPS is designed to be easy to modify or extend with new capabilities, such as new force fields, atom types, boundary conditions, or diagnostics. See this section for more details.

The current version of LAMMPS is written in C++. Earlier versions were written in F77 and F90. See this section for more information on different versions. All versions can be downloaded from the LAMMPS WWW Site.

LAMMPS was originally developed under a US Department of Energy CRADA (Cooperative Research and Development Agreement) between two DOE labs and 3 companies. It is distributed by Sandia National Labs. See this section for more information on LAMMPS funding and individuals who have contributed to LAMMPS.

In the most general sense, LAMMPS integrates Newton's equations of motion for collections of atoms, molecules, or macroscopic particles that interact via short− or long−range forces with a variety of initial and/or boundary conditions. For computational efficiency LAMMPS uses neighbor lists to keep track of nearby particles. The lists are optimized for systems with particles that are repulsive at short distances, so that the local density of particles never becomes too large. On parallel machines, LAMMPS uses spatial−decomposition techniques to partition the simulation domain into small 3d sub−domains, one of which is assigned to each processor. Processors communicate and store "ghost" atom information for atoms that border their sub−domain. LAMMPS is most efficient (in a parallel sense) for systems whose particles fill a 3d rectangular box with roughly uniform density.

Papers with technical details of the algorithms used in LAMMPS are listed in this section.

1.2 LAMMPS features

This section highlights LAMMPS features, with pointers to specific commands which give more details. If LAMMPS doesn't have your favorite interatomic potential, boundary condition, or atom type, see this section, which describes how you can add it to LAMMPS.

General features

- runs on a single processor or in parallel
- distributed−memory message−passing parallelism (MPI)
- spatial−decomposition of simulation domain for parallelism
- open−source distribution
- highly portable C++
- optional libraries used: MPI and single−processor FFT
- easy to extend with new features and functionality
- runs from an input script
- syntax for defining and using variables and formulas
- syntax for looping over runs and breaking out of loops
- run one or multiple simulations simultaneously (in parallel) from one script
- build as library, invoke LAMMPS thru library interface or provided Python wrapper
- couple with other codes: LAMMPS calls other code, other code calls LAMMPS, umbrella code calls both

Particle and model types

(atom style command)

- atoms
- coarse−grained particles (e.g. bead−spring polymers)
- united−atom polymers or organic molecules
- all−atom polymers, organic molecules, proteins, DNA
- metals
- granular materials
- coarse−grained mesoscale models
- extended spherical and ellipsoidal particles
- point dipolar particles
- rigid collections of particles
- hybrid combinations of these

Force fields

(pair style, bond style, angle style, dihedral style, improper style, kspace style commands)

- pairwise potentials: Lennard−Jones, Buckingham, Morse, Born−Mayer−Huggins, Yukawa, soft, class 2 (COMPASS), hydrogen bond, tabulated
- charged pairwise potentials: Coulombic, point−dipole
- manybody potentials: EAM, Finnis/Sinclair EAM, modified EAM (MEAM), embedded ion method (EIM), ADP, Stillinger−Weber, Tersoff, REBO, AIREBO, ReaxFF, COMB
- electron force field (eFF, AWPMD)
- coarse−grained potentials: DPD, GayBerne, REsquared, colloidal, DLVO

- mesoscopic potentials: granular, Peridynamics
- bond potentials: harmonic, FENE, Morse, nonlinear, class 2, quartic (breakable)
- angle potentials: harmonic, CHARMM, cosine, cosine/squared, cosine/periodic, class 2 (COMPASS)
- dihedral potentials: harmonic, CHARMM, multi−harmonic, helix, class 2 (COMPASS), OPLS
- improper potentials: harmonic, cvff, umbrella, class 2 (COMPASS)
- polymer potentials: all−atom, united−atom, bead−spring, breakable
- water potentials: TIP3P, TIP4P, SPC
- implicit solvent potentials: hydrodynamic lubrication, Debye
- long−range Coulombics and dispersion: Ewald, PPPM (similar to particle−mesh Ewald), Ewald/N for long−range Lennard−Jones
- force−field compatibility with common CHARMM, AMBER, DREIDING, OPLS, GROMACS, COMPASS options
- handful of GPU−enabled pair styles

hybrid potentials: multiple pair, bond, angle, dihedral, improper potentials can be used in one simulation overlaid potentials: superposition of multiple pair potentials

Atom creation

(read_data, lattice, create_atoms, delete_atoms, displace_atoms, replicate commands)

- read in atom coords from files
- create atoms on one or more lattices (e.g. grain boundaries)
- delete geometric or logical groups of atoms (e.g. voids)
- replicate existing atoms multiple times
- displace atoms

Ensembles, constraints, and boundary conditions

(fix command)

- 2d or 3d systems
- orthogonal or non−orthogonal (triclinic symmetry) simulation domains
- constant NVE, NVT, NPT, NPH, Parinello/Rahman integrators
- thermostatting options for groups and geometric regions of atoms
- pressure control via Nose/Hoover or Berendsen barostatting in 1 to 3 dimensions
- simulation box deformation (tensile and shear)
- harmonic (umbrella) constraint forces
- rigid body constraints
- SHAKE bond and angle constraints
- bond breaking, formation, swapping
- walls of various kinds
- non−equilibrium molecular dynamics (NEMD)
- variety of additional boundary conditions and constraints

Integrators

(run, run_style, minimize commands)

- velocity−Verlet integrator
- Brownian dynamics
- rigid body integration

- energy minimization via conjugate gradient or steepest descent relaxation
- rRESPA hierarchical timestepping

Diagnostics

- see the various flavors of the fix and compute commands

Output

(dump, restart commands)

- log file of thermodynamic info
- text dump files of atom coords, velocities, other per−atom quantities
- binary restart files
- parallel I/O of dump and restart files
- per−atom quantities (energy, stress, centro−symmetry parameter, CNA, etc)
- user−defined system−wide (log file) or per−atom (dump file) calculations
- spatial and time averaging of per−atom quantities
- time averaging of system−wide quantities
- atom snapshots in native, XYZ, XTC, DCD, CFG formats

Multi−replica models

nudged elastic band parallel replica dynamics temperature accelerated dynamics parallel tempering

Pre− and post−processing

- Various pre− and post−processing serial tools are packaged with LAMMPS; see these doc pages.
- Our group has also written and released a separate toolkit called Pizza.py which provides tools for doing setup, analysis, plotting, and visualization for LAMMPS simulations. Pizza.py is written in Python and is available for download from the Pizza.py WWW site.

Specialized features

These are LAMMPS capabilities which you may not think of as typical molecular dynamics options:

- stochastic rotation dynamics (SRD)
- real−time visualization and interactive MD
- atom−to−continuum coupling with finite elements
- coupled rigid body integration via the POEMS library
- grand canonical Monte Carlo insertions/deletions
- Direct Simulation Monte Carlo for low−density fluids
- Peridynamics mesoscale modeling
- targeted and steered molecular dynamics
- two−temperature electron model

1.3 LAMMPS non−features

LAMMPS is designed to efficiently compute Newton's equations of motion for a system of interacting particles. Many of the tools needed to pre− and post−process the data for such simulations are not included in the LAMMPS kernel for several reasons:

- the desire to keep LAMMPS simple
- they are not parallel operations
- other codes already do them
- limited development resources

Specifically, LAMMPS itself does not:

- run thru a GUI
- build molecular systems
- assign force−field coefficients automagically
- perform sophisticated analyses of your MD simulation
- visualize your MD simulation
- plot your output data

A few tools for pre− and post−processing tasks are provided as part of the LAMMPS package; they are described in this section. However, many people use other codes or write their own tools for these tasks.

As noted above, our group has also written and released a separate toolkit called Pizza.py which addresses some of the listed bullets. It provides tools for doing setup, analysis, plotting, and visualization for LAMMPS simulations. Pizza.py is written in Python and is available for download from the Pizza.py WWW site.

LAMMPS requires as input a list of initial atom coordinates and types, molecular topology information, and force−field coefficients assigned to all atoms and bonds. LAMMPS will not build molecular systems and assign force−field parameters for you.

For atomic systems LAMMPS provides a create_atoms command which places atoms on solid−state lattices (fcc, bcc, user−defined, etc). Assigning small numbers of force field coefficients can be done via the pair coeff, bond coeff, angle coeff, etc commands. For molecular systems or more complicated simulation geometries, users typically use another code as a builder and convert its output to LAMMPS input format, or write their own code to generate atom coordinate and molecular topology for LAMMPS to read in.

For complicated molecular systems (e.g. a protein), a multitude of topology information and hundreds of force−field coefficients must typically be specified. We suggest you use a program like CHARMM or AMBER or other molecular builders to setup such problems and dump its information to a file. You can then reformat the file as LAMMPS input. Some of the tools in this section can assist in this process.

Similarly, LAMMPS creates output files in a simple format. Most users post−process these files with their own analysis tools or re−format them for input into other programs, including visualization packages. If you are convinced you need to compute something on−the−fly as LAMMPS runs, see this section for a discussion of how you can use the dump and compute and fix commands to print out data of your choosing. Keep in mind that complicated computations can slow down the molecular dynamics timestepping, particularly if the computations are not parallel, so it is often better to leave such analysis to post−processing codes.

A very simple (yet fast) visualizer is provided with the LAMMPS package − see the xmovie tool in this section. It creates xyz projection views of atomic coordinates and animates them. We find it very useful for debugging purposes. For high−quality visualization we recommend the following packages:

- VMD
- AtomEye
- PyMol
- Raster3d
- RasMol

Other features that LAMMPS does not yet (and may never) support are discussed in this section.

Finally, these are freely−available molecular dynamics codes, most of them parallel, which may be well−suited to the problems you want to model. They can also be used in conjunction with LAMMPS to perform complementary modeling tasks.

- CHARMM
- AMBER
- NAMD
- NWCHEM
- DL_POLY
- Tinker

CHARMM, AMBER, NAMD, NWCHEM, and Tinker are designed primarily for modeling biological molecules. CHARMM and AMBER use atom−decomposition (replicated−data) strategies for parallelism; NAMD and NWCHEM use spatial−decomposition approaches, similar to LAMMPS. Tinker is a serial code. DL_POLY includes potentials for a variety of biological and non−biological materials; both a replicated−data and spatial−decomposition version exist.

1.4 Open source distribution

LAMMPS comes with no warranty of any kind. As each source file states in its header, it is a copyrighted code that is distributed free−of− charge, under the terms of the GNU Public License (GPL). This is often referred to as open−source distribution − see www.gnu.org or www.opensource.org for more details. The legal text of the GPL is in the LICENSE file that is included in the LAMMPS distribution.

Here is a summary of what the GPL means for LAMMPS users:

(1) Anyone is free to use, modify, or extend LAMMPS in any way they choose, including for commercial purposes.

(2) If you distribute a modified version of LAMMPS, it must remain open−source, meaning you distribute it under the terms of the GPL. You should clearly annotate such a code as a derivative version of LAMMPS.

(3) If you release any code that includes LAMMPS source code, then it must also be open−sourced, meaning you distribute it under the terms of the GPL.

(4) If you give LAMMPS files to someone else, the GPL LICENSE file and source file headers (including the copyright and GPL notices) should remain part of the code.

In the spirit of an open−source code, these are various ways you can contribute to making LAMMPS better. You can send email to the developers on any of these items.

- Point prospective users to the LAMMPS WWW Site. Mention it in talks or link to it from your WWW site.
- If you find an error or omission in this manual or on the LAMMPS WWW Site, or have a suggestion for something to clarify or include, send an email to the developers.
- If you find a bug, this section describes how to report it.
- If you publish a paper using LAMMPS results, send the citation (and any cool pictures or movies if you like) to add to the Publications, Pictures, and Movies pages of the LAMMPS WWW Site, with links and attributions back to you.

- Create a new Makefile.machine that can be added to the src/MAKE directory.
- The tools sub−directory of the LAMMPS distribution has various stand−alone codes for pre− and post−processing of LAMMPS data. More details are given in this section. If you write a new tool that users will find useful, it can be added to the LAMMPS distribution.
- LAMMPS is designed to be easy to extend with new code for features like potentials, boundary conditions, diagnostic computations, etc. This section gives details. If you add a feature of general interest, it can be added to the LAMMPS distribution.
- The Benchmark page of the LAMMPS WWW Site lists LAMMPS performance on various platforms. The files needed to run the benchmarks are part of the LAMMPS distribution. If your machine is sufficiently different from those listed, your timing data can be added to the page.
- You can send feedback for the User Comments page of the LAMMPS WWW Site. It might be added to the page. No promises.
- Cash. Small denominations, unmarked bills preferred. Paper sack OK. Leave on desk. VISA also accepted. Chocolate chip cookies encouraged.

1.5 Acknowledgments and citations

LAMMPS development has been funded by the US Department of Energy (DOE), through its CRADA, LDRD, ASCI, and Genomes−to−Life programs and its OASCR and OBER offices.

Specifically, work on the latest version was funded in part by the US Department of Energy's Genomics:GTL program (www.doegenomestolife.org) under the project, "Carbon Sequestration in Synechococcus Sp.: From Molecular Machines to Hierarchical Modeling".

The following papers describe the parallel algorithms used in LAMMPS.

S. J. Plimpton, **Fast Parallel Algorithms for Short−Range Molecular Dynamics**, J Comp Phys, 117, 1−19 (1995).

S. J. Plimpton, R. Pollock, M. Stevens, **Particle−Mesh Ewald and rRESPA for Parallel Molecular Dynamics Simulations**, in Proc of the Eighth SIAM Conference on Parallel Processing for Scientific Computing, Minneapolis, MN (March 1997).

If you use LAMMPS results in your published work, please cite the J Comp Phys reference and include a pointer to the LAMMPS WWW Site (http://lammps.sandia.gov).

If you send is information about your publication, we'll be pleased to add it to the Publications page of the LAMMPS WWW Site. Ditto for a picture or movie for the Pictures or Movies pages.

The core group of LAMMPS developers is at Sandia National Labs. They include Steve Plimpton, Paul Crozier, and Aidan Thompson and can be contacted via email: sjplimp, pscrozi, athomps at sandia.gov.

Here are various folks who have made significant contributions to features in LAMMPS. The most recent contributions are at the top of the list.

pppm GPU single and double	Mike Brown (ORNL)
pair_style lj/cut/expand	Inderaj Bains (NVIDIA)
temperature accelerated dynamics (TAD)	Aidan Thompson (Sandia)
pair reax/c and fix qeq/reax	Metin Aktulga (Purdue, now LBNL)
DREIDING force field, pair_style hbond/dreiding, etc	Tod Pascal (CalTech)

fix adapt and compute ti for thermodynamic integreation for free energies	Sai Jayaraman (Sandia)
pair born and pair gauss	Sai Jayaraman (Sandia)
stochastic rotation dynamics (SRD) via fix srd	Jemery Lechman (Sandia) and Pieter in 't Veld (BASF)
ipp Perl script tool	Reese Jones (Sandia)
eam_database and createatoms tools	Xiaowang Zhou (Sandia)
electron force field (eFF)	Andres Jaramillo−Botero and Julius Su (Caltech)
embedded ion method (EIM) potential	Xiaowang Zhou (Sandia)
COMB potential with charge equilibration	Tzu−Ray Shan (U Florida)
fix ave/correlate	Benoit Leblanc, Dave Rigby, Paul Saxe (Materials Design) and Reese Jones (Sandia)
pair_style peri/lps	Mike Parks (Sandia)
fix msst	Lawrence Fried (LLNL), Evan Reed (LLNL, Stanford)
thermo_style custom tpcpu &spcpu keywords	Axel Kohlmeyer (Temple U)
fix rigid/nve, fix rigid/nvt	Tony Sheh and Trung Dac Nguyen (U Michigan)
public SVN &Git repositories for LAMMPS	Axel Kohlmeyer (Temple U) and Bill Goldman (Sandia)
fix nvt, fix nph, fix npt, Parinello/Rahman dynamics, fix box/relax	Aidan Thompson (Sandia)
compute heat/flux	German Samolyuk (ORNL) and Mario Pinto (Computational Research Lab, Pune, India)
pair yukawa/colloid	Randy Schunk (Sandia)
fix wall/colloid	Jeremy Lechman (Sandia)
pair_style dsmc for Direct Simulation Monte Carlo (DSMC) modeling	Paul Crozier (Sandia)
fix imd for real−time viz and interactive MD	Axel Kohlmeyer (Temple Univ)
concentration−dependent EAM potential	Alexander Stukowski (Technical University of Darmstadt)
parallel replica dymamics (PRD)	Mike Brown (Sandia)
min_style hftn	Todd Plantenga (Sandia)
fix atc	Reese Jones, Jon Zimmerman, Jeremy Templeton (Sandia)
dump cfg	Liang Wan (Chinese Academy of Sciences)
fix nvt with Nose/Hoover chains	Andy Ballard (U Maryland)
pair_style lj/cut/gpu, pair_style gayberne/gpu	Mike Brown (Sandia)
pair_style lj96/cut, bond_style table, angle_style table	Chuanfu Luo
fix langevin tally	Carolyn Phillips (U Michigan)
compute heat/flux for Green−Kubo	Reese Jones (Sandia), Philip Howell (Siemens), Vikas Varsney (AFRL)
region cone	Pim Schravendijk
fix reax/bonds	Aidan Thompson (Sandia)
pair born/coul/long	Ahmed Ismail (Sandia)
fix ttm	Paul Crozier (Sandia) and Carolyn Phillips (U Michigan)
fix box/relax	Aidan Thompson and David Olmsted (Sandia)
ReaxFF potential	Aidan Thompson (Sandia) and Hansohl Cho (MIT)
compute cna/atom	Wan Liang (Chinese Academy of Sciences)

Tersoff/ZBL potential	Dave Farrell (Northwestern U)
peridynamics	Mike Parks (Sandia)
fix smd for steered MD	Axel Kohlmeyer (U Penn)
GROMACS pair potentials	Mark Stevens (Sandia)
lmp2vmd tool	Axel Kohlmeyer (U Penn)
compute group/group	Naveen Michaud−Agrawal (Johns Hopkins U)
CG−CMM user package for coarse−graining	Axel Kohlmeyer (U Penn)
cosine/delta angle potential	Axel Kohlmeyer (U Penn)
VIM editor add−ons for LAMMPS input scripts	Gerolf Ziegenhain
pair lubricate	Randy Schunk (Sandia)
compute ackland/atom	Gerolf Zeigenhain
kspace_style ewald/n, pair_style lj/coul, pair_style buck/coul	Pieter in 't Veld (Sandia)
AIREBO bond−order potential	Ase Henry (MIT)
making LAMMPS a true "object" that can be instantiated multiple times, e.g. as a library	Ben FrantzDale (RPI)
pymol_asphere viz tool	Mike Brown (Sandia)
NEMD SLLOD integration	Pieter in 't Veld (Sandia)
tensile and shear deformations	Pieter in 't Veld (Sandia)
GayBerne potential	Mike Brown (Sandia)
ellipsoidal particles	Mike Brown (Sandia)
colloid potentials	Pieter in 't Veld (Sandia)
fix heat	Paul Crozier and Ed Webb (Sandia)
neighbor multi and communicate multi	Pieter in 't Veld (Sandia)
MATLAB post−processing scripts	Arun Subramaniyan (Purdue)
triclinic (non−orthogonal) simulation domains	Pieter in 't Veld (Sandia)
thermo_extract tool	Vikas Varshney (Wright Patterson AFB)
fix ave/time and fix ave/spatial	Pieter in 't Veld (Sandia)
MEAM potential	Greg Wagner (Sandia)
optimized pair potentials for lj/cut, charmm/long, eam, morse	James Fischer (High Performance Technologies), David Richie and Vincent Natoli (Stone Ridge Technologies)
fix wall/lj126	Mark Stevens (Sandia)
Stillinger−Weber and Tersoff potentials	Aidan Thompson and Xiaowang Zhou (Sandia)
region prism	Pieter in 't Veld (Sandia)
LJ tail corrections for energy/pressure	Paul Crozier (Sandia)
fix momentum and recenter	Naveen Michaud−Agrawal (Johns Hopkins U)
multi−letter variable names	Naveen Michaud−Agrawal (Johns Hopkins U)
OPLS dihedral potential	Mark Stevens (Sandia)
POEMS coupled rigid body integrator	Rudranarayan Mukherjee (RPI)
faster pair hybrid potential	James Fischer (High Performance Technologies, Inc), Vincent Natoli and David Richie (Stone Ridge Technology)
breakable bond quartic potential	Chris Lorenz and Mark Stevens (Sandia)
DCD and XTC dump styles	Naveen Michaud−Agrawal (Johns Hopkins U)

grain boundary orientation fix	Koenraad Janssens and David Olmsted (Sandia)
lj/smooth pair potential	Craig Maloney (UCSB)
radius−of−gyration spring fix	Naveen Michaud−Agrawal (Johns Hopkins U) and Paul Crozier (Sandia)
self spring fix	Naveen Michaud−Agrawal (Johns Hopkins U)
EAM CoAl and AlCu potentials	Kwang−Reoul Lee (KIST, Korea)
cosine/squared angle potential	Naveen Michaud−Agrawal (Johns Hopkins U)
helix dihedral potential	Naveen Michaud−Agrawal (Johns Hopkins U) and Mark Stevens (Sandia)
Finnis/Sinclair EAM	Tim Lau (MIT)
dissipative particle dynamics (DPD) potentials	Kurt Smith (U Pitt) and Frank van Swol (Sandia)
TIP4P potential (4−site water)	Ahmed Ismail and Amalie Frischknecht (Sandia)
uniaxial strain fix	Carsten Svaneborg (Max Planck Institute)
thermodynamics enhanced by fix quantities	Aidan Thompson (Sandia)
compressed dump files	Erik Luijten (U Illinois)
cylindrical indenter fix	Ravi Agrawal (Northwestern U)
electric field fix	Christina Payne (Vanderbilt U)
AMBER LAMMPS tool	Keir Novik (Univ College London) and Vikas Varshney (U Akron)
CHARMM LAMMPS tool	Pieter in 't Veld and Paul Crozier (Sandia)
Morse bond potential	Jeff Greathouse (Sandia)
radial distribution functions	Paul Crozier &Jeff Greathouse (Sandia)
force tables for long−range Coulombics	Paul Crozier (Sandia)
targeted molecular dynamics (TMD)	Paul Crozier (Sandia) and Christian Burisch (Bochum University, Germany)
FFT support for SGI SCSL (Altix)	Jim Shepherd (Ga Tech)
lmp2cfg and lmp2traj tools	Ara Kooser, Jeff Greathouse, Andrey Kalinichev (Sandia)
parallel tempering	Mark Sears (Sandia)
embedded atom method (EAM) potential	Stephen Foiles (Sandia)
multi−harmonic dihedral potential	Mathias Puetz (Sandia)
granular force fields and BC	Leo Silbert &Gary Grest (Sandia)
2d Ewald/PPPM	Paul Crozier (Sandia)
CHARMM force fields	Paul Crozier (Sandia)
msi2lmp tool	Steve Lustig (Dupont), Mike Peachey &John Carpenter (Cray)
HTFN energy minimizer	Todd Plantenga (Sandia)
class 2 force fields	Eric Simon (Cray)
NVT/NPT integrators	Mark Stevens (Sandia)
rRESPA	Mark Stevens &Paul Crozier (Sandia)
Ewald and PPPM solvers	Roy Pollock (LLNL)

Other CRADA partners involved in the design and testing of LAMMPS were

- John Carpenter (Mayo Clinic, formerly at Cray Research)
- Terry Stouch (Lexicon Pharmaceuticals, formerly at Bristol Myers Squibb)
- Steve Lustig (Dupont)

- Jim Belak (LLNL)

2. Getting Started

This section describes how to build and run LAMMPS, for both new and experienced users.

2.1 What's in the LAMMPS distribution
2.2 Making LAMMPS
2.3 Making LAMMPS with optional packages
2.4 Building LAMMPS as a library
2.5 Running LAMMPS
2.6 Command−line options
2.7 Screen output
2.8 Tips for users of previous versions

2.1 What's in the LAMMPS distribution

When you download LAMMPS you will need to unzip and untar the downloaded file with the following commands, after placing the file in an appropriate directory.

```
gunzip lammps*.tar.gz
tar xvf lammps*.tar
```

This will create a LAMMPS directory containing two files and several sub−directories:

README	text file
LICENSE	the GNU General Public License (GPL)
bench	benchmark problems
couple	code coupling examples, using LAMMPS as a library
doc	documentation
examples	simple test problems
potentials	embedded atom method (EAM) potential files
src	source files
tools	pre− and post−processing tools

If you download one of the Windows executables from the download page, then you just get a single file:

```
lmp_windows.exe
```

Skip to the Running LAMMPS sections for info on how to launch these executables on a Windows box.

The Windows executables for serial or parallel only include certain packages and bug−fixes/upgrades listed on this page up to a certain date, as stated on the download page. If you want something with more packages or that is more current, you'll have to download the source tarball and build it yourself from source code using Microsoft Visual Studio, as described in the next section.

2.2 Making LAMMPS

This section has the following sub−sections:

- Read this first

- Steps to build a LAMMPS executable
- Common errors that can occur when making LAMMPS
- Additional build tips
- Building for a Mac
- Building for Windows

Read this first:

Building LAMMPS can be non−trivial. You may need to edit a makefile, there are compiler options to consider, additional libraries can be used (MPI, FFT, JPEG), LAMMPS packages may be included or excluded, some of these packages use auxiliary libraries which need to be pre−built, etc.

Please read this section carefully. If you are not comfortable with makefiles, or building codes on a Unix platform, or running an MPI job on your machine, please find a local expert to help you. Many compiling, linking, and run problems that users have are not really LAMMPS issues − they are peculiar to the user's system, compilers, libraries, etc. Such questions are better answered by a local expert.

If you have a build problem that you are convinced is a LAMMPS issue (e.g. the compiler complains about a line of LAMMPS source code), then please send an email to the developers.

If you succeed in building LAMMPS on a new kind of machine, for which there isn't a similar Makefile for in the src/MAKE directory, send it to the developers and we'll include it in future LAMMPS releases.

Steps to build a LAMMPS executable:

Step 0

The src directory contains the C++ source and header files for LAMMPS. It also contains a top−level Makefile and a MAKE sub−directory with low−level Makefile.* files for many machines. From within the src directory, type "make" or "gmake". You should see a list of available choices. If one of those is the machine and options you want, you can type a command like:

```
make linux
or
gmake mac
```

Note that on a multi−processor or multi−core platform you can launch a parallel make, by using the "−j" switch with the make command, which will build LAMMPS more quickly.

If you get no errors and an executable like lmp_linux or lmp_mac is produced, you're done; it's your lucky day.

Note that by default only a few of LAMMPS optional pacakges are installed. To build LAMMPS with optional packages, see this section below.

Step 1

If Step 0 did not work, you will need to create a low−level Makefile for your machine, like Makefile.foo. You should make a copy of an existing src/MAKE/Makefile.* as a starting point. The only portions of the file you need to edit are the first line, the "compiler/linker settings" section, and the "LAMMPS−specific settings" section.

Step 2

Change the first line of src/MAKE/Makefile.foo to list the word "foo" after the "#", and whatever other options it will set. This is the line you will see if you just type "make".

Step 3

The "compiler/linker settings" section lists compiler and linker settings for your C++ compiler, including optimization flags. You can use g++, the open−source GNU compiler, which is available on all Unix systems. You can also use mpicc which will typically be available if MPI is installed on your system, though you should check which actual compiler it wraps. Vendor compilers often produce faster code. On boxes with Intel CPUs, we suggest using the commercial Intel icc compiler, which can be downloaded from Intel's compiler site.

If building a C++ code on your machine requires additional libraries, then you should list them as part of the LIB variable.

The DEPFLAGS setting is what triggers the C++ compiler to create a dependency list for a source file. This speeds re−compilation when source (*.cpp) or header (*.h) files are edited. Some compilers do not support dependency file creation, or may use a different switch than −D. GNU g++ works with −D. If your compiler can't create dependency files, then you'll need to create a Makefile.foo patterned after Makefile.storm, which uses different rules that do not involve dependency files. Note that when you build LAMMPS for the first time on a new platform, a long list of *.d files will be printed out rapidly. This is not an error; it is the Makefile doing its normal creation of dependencies.

Step 4

The "system−specific settings" section has several parts. Note that if you change any −D setting in this section, you should do a full re−compile, after typing "make clean" (which will describe different clean options).

The LMP_INC variable is used to include options that turn on ifdefs within the LAMMPS code. The options that are currently recogized are:

- −DLAMMPS_GZIP
- −DLAMMPS_JPEG
- −DLAMMPS_XDR
- −DLAMMPS_SMALLBIG
- −DLAMMPS_BIGBIG
- −DLAMMPS_SMALLSMALL
- −DLAMMPS_LONGLONG_TO_LONG
- −DPACK_ARRAY
- −DPACK_POINTER
- −DPACK_MEMCPY

The read_data and dump commands will read/write gzipped files if you compile with −DLAMMPS_GZIP. It requires that your Unix support the "popen" command.

If you use −DLAMMPS_JPEG, the dump image command will be able to write out JPEG image files. If not, it will only be able to write out text−based PPM image files. For JPEG files, you must also link LAMMPS with a JPEG library, as described below.

If you use −DLAMMPS_XDR, the build will include XDR compatibility files for doing particle dumps in XTC format. This is only necessary if your platform does have its own XDR files available. See the Restrictions section of the dump command for details.

Use at most one of the −DLAMMPS_SMALLBIG, −DLAMMPS_BIGBIG, −D−DLAMMPS_SMALLSMALL settings. The default is −DLAMMPS_SMALLBIG. These refer to use of 4−byte (small) vs 8−byte (big) integers within LAMMPS, as described in src/lmptype.h. The only reason to use the BIGBIG setting is to enable simulation of huge molecular systems with more than 2 billion atoms. The only reason to use the SMALLSMALL setting is if your machine does not support 64−bit integers.

The −DLAMMPS_LONGLONG_TO_LONG setting may be needed if your system or MPI version does not recognize "long long" data types. In this case a "long" data type is likely already 64−bits, in which case this setting will convert to that data type.

Using one of the −DPACK_ARRAY, −DPACK_POINTER, and −DPACK_MEMCPY options can make for faster parallel FFTs (in the PPPM solver) on some platforms. The −DPACK_ARRAY setting is the default. See the kspace_style command for info about PPPM. See Step 6 below for info about building LAMMPS with an FFT library.

Step 5

The 3 MPI variables are used to specify an MPI library to build LAMMPS with.

If you want LAMMPS to run in parallel, you must have an MPI library installed on your platform. If you use an MPI−wrapped compiler, such as "mpicc" to build LAMMPS, you should be able to leave these 3 variables blank; the MPI wrapper knows where to find the needed files. If not, and MPI is installed on your system in the usual place (under /usr/local), you also may not need to specify these 3 variables. On some large parallel machines which use "modules" for their compile/link environements, you may simply need to include the correct module in your build environment. Or the parallel machine may have a vendor−provided MPI which the compiler has no trouble finding.

Failing this, with these 3 variables you can specify where the mpi.h file (MPI_INC) and the MPI library file (MPI_PATH) are found and the name of the library file (MPI_LIB).

If you are installing MPI yourself, we recommend Argonne's MPICH 1.2 or 2.0 or OpenMPI. MPICH can be downloaded from the Argonne MPI site. OpenMPI can be downloaded the OpenMPI site. LAM MPI should also work. If you are running on a big parallel platform, your system people or the vendor should have already installed a version of MPI, which will be faster than MPICH or OpenMPI or LAM, so find out how to build and link with it. If you use MPICH or OpenMPI or LAM, you will have to configure and build it for your platform. The MPI configure script should have compiler options to enable you to use the same compiler you are using for the LAMMPS build, which can avoid problems that can arise when linking LAMMPS to the MPI library.

If you just want to run LAMMPS on a single processor, you can use the dummy MPI library provided in src/STUBS, since you don't need a true MPI library installed on your system. See the src/MAKE/Makefile.serial file for how to specify the 3 MPI variables in this case. You will also need to build the STUBS library for your platform before making LAMMPS itself. From the src directory, type "make stubs", or from the STUBS dir, type "make" and it should create a libmpi.a suitable for linking to LAMMPS. If this build fails, you will need to edit the STUBS/Makefile for your platform.

The file STUBS/mpi.cpp provides a CPU timer function called MPI_Wtime() that calls gettimeofday() . If your system doesn't support gettimeofday() , you'll need to insert code to call another timer. Note that the ANSI−standard function clock() rolls over after an hour or so, and is therefore insufficient for timing long LAMMPS simulations.

Step 6

The 3 FFT variables allow you to specify an FFT library which LAMMPS uses (for performing 1d FFTs) when running the particle−particle particle−mesh (PPPM) option for long−range Coulombics via the kspace_style command.

LAMMPS supports various open−source or vendor−supplied FFT libraries for this purpose. If you leave these 3 variables blank, LAMMPS will use the open−source KISS FFT library, which is included in the LAMMPS distribution. This library is portable to all platforms and for typical LAMMPS simulations is almost as fast as FFTW or vendor optimized libraries. If you are not including the KSPACE package in your build, you can also leave the 3 variables blank.

Otherwise, select which kinds of FFTs to use as part of the FFT_INC setting by a switch of the form −DFFT_XXX. Recommended values for XXX are: MKL, SCSL, FFTW2, and FFTW3. Legacy options are: INTEL, SGI, ACML, and T3E. For backward compatability, using −DFFT_FFTW will use the FFTW2 library. Using −DFFT_NONE will use the KISS library described above.

You may also need to set the FFT_INC, FFT_PATH, and FFT_LIB variables, so the compiler and linker can find the needed FFT header and library files. Note that on some large parallel machines which use "modules" for their compile/link environements, you may simply need to include the correct module in your build environment. Or the parallel machine may have a vendor−provided FFT library which the compiler has no trouble finding.

FFTW is a fast, portable library that should also work on any platform. You can download it from www.fftw.org. Both the legacy version 2.1.X and the newer 3.X versions are supported as −DFFT_FFTW2 or −DFFT_FFTW3. Building FFTW for your box should be as simple as ./configure; make. Note that on some platforms FFTW2 has been pre−installed, and uses renamed files indicating the precision it was compiled with, e.g. sfftw.h, or dfftw.h instead of fftw.h. In this case, you can specify an additional define variable for FFT_INC called −DFFTW2_SIZE, which will select the correct include file. In this case, for FFT_LIB you must also manually specify the correct library, namely −lsfftw or −ldfftw.

The FFT_INC variable also allows for a −DFFT_SINGLE setting that will use single−precision FFTs with PPPM, which can speed−up long−range calulations, particularly in parallel or on GPUs. Fourier transform and related PPPM operations are somewhat insensitive to floating point truncation errors and thus do not always need to be performed in double precision. Using the −DFFT_SINGLE setting trades off a little accuracy for reduced memory use and parallel communication costs for transposing 3d FFT data. Note that single precision FFTs have only been tested with the FFTW3, FFTW2, MKL, and KISS FFT packages.

Step 7

The 3 JPG variables allow you to specify a JPEG library which LAMMPS uses when writing out JPEG files via the dump image command. These can be left blank if you do not use the −DLAMMPS_JPEG switch discussed above in Step 4, since in that case JPEG output will be disabled.

A standard JPEG library usually goes by the name libjpeg.a and has an associated header file jpeglib.h. Whichever JPEG library you have on your platform, you'll need to set the appropriate JPG_INC, JPG_PATH, and JPG_LIB variables, so that the compiler and linker can find it.

As before, if these header and library files are in the usual place on your machine, you may not need to set these variables.

Step 8

Note that by default only a few of LAMMPS optional pacakges are installed. To build LAMMPS with optional packages, see this section below, before proceeding to Step 9.

Step 9

That's it. Once you have a correct Makefile.foo, you have installed the optional LAMMPS packages you want to include in your build, and you have pre−built any other needed libraries (e.g. MPI, FFT, package libraries), all you need to do from the src directory is type something like this:

```
make foo
or
gmake foo
```

You should get the executable lmp_foo when the build is complete.

Errors that can occur when making LAMMPS:

IMPORTANT NOTE: If an error occurs when building LAMMPS, the compiler or linker will state very explicitly what the problem is. The error message should give you a hint as to which of the steps above has failed, and what you need to do in order to fix it. Building a code with a Makefile is a very logical process. The compiler and linker need to find the appropriate files and those files need to be compatible with LAMMPS source files. When a make fails, there is usually a very simple reason, which you or a local expert will need to fix.

Here are two non−obvious errors that can occur:

(1) If the make command breaks immediately with errors that indicate it can't find files with a "*" in their names, this can be because your machine's native make doesn't support wildcard expansion in a makefile. Try gmake instead of make. If that doesn't work, try using a −f switch with your make command to use a pre−generated Makefile.list which explicitly lists all the needed files, e.g.

```
make makelist
make −f Makefile.list linux
gmake −f Makefile.list mac
```

The first "make" command will create a current Makefile.list with all the file names in your src dir. The 2nd "make" command (make or gmake) will use it to build LAMMPS. Note that you should include/exclude any desired optional packages before using the "make makelist" command.

(2) If you get an error that says something like 'identifier "atoll" is undefined', then your machine does not support "long long" integers. Try using the −DLAMMPS_LONGLONG_TO_LONG setting described above in Step 4.

Additional build tips:

(1) Building LAMMPS for multiple platforms.

You can make LAMMPS for multiple platforms from the same src directory. Each target creates its own object sub−directory called Obj_name where it stores the system−specific *.o files.

(2) Cleaning up.

Typing "make clean−all" or "make clean−foo" will delete *.o object files created when LAMMPS is built, for either all builds or for a particular machine.

(3) Changing the size limits in src/lmptype.h

If you are running a very large problem (billions of atoms or more) and get a run−time error about the system being too big, either on a per−processor basis or in total size, then you may need to change one or more settings in src/lmptype.h and re−compile LAMMPS.

As the documentation in that file explains, you have basically two choices to make:

- set the data type size of integer atom IDs to 4 or 8 bytes
- set the data type size of integers that store the total system size to 4 or 8 bytes

The default for atom IDs is 4−byte integers since there is a memory and communication cost for 8−byte integers. Non−molecular problems do not need atom IDs so this does not restrict their size. Molecular problems (which use IDs to define molecular topology), are limited to about 2 billion atoms (2^31) with 4−byte IDs. With 8−byte IDs they are effectively unlimited in size (2^63).

The default for total system size quantities (like the number of atoms or timesteps) is 8−byte integers by default which is effectively unlimited in size (2^63). If your system or MPI implementation does not support 8−byte integers, an error will be generated, and you will need to set "bigint" to 4−byte integers. This restricts your total system size to about 2 billion atoms or timesteps (2^31).

Note that in src/lmptype.h there are also settings for the MPI data types associated with the integers that store atom IDs and total system sizes, which need to be set consistent with the associated C data types.

In all cases, the size of problem that can be run on a per−processor basis is limited by 4−byte integer storage to about 2 billion atoms per processor (2^31), which should not normally be a restriction since such a problem would have a huge per−processor memory footprint due to neighbor lists and would run very slowly in terms of CPU secs/timestep.

Building for a Mac:

OS X is BSD Unix, so it should just work. See the Makefile.mac file.

Building for Windows:

The LAMMPS download page has an option to download both a serial and parallel pre−built Windows exeutable. See the Running LAMMPS section for instructions for running these executables on a Windows box.

If the pre−built executable doesn't have the options you want, then you can build LAMMPS from its source files on a Windows box. One way to do this is install and use cygwin to build LAMMPS with a standard Linus make, just as you would on any Linux box; see src/MAKE/Makefile.cygwin.

There is a also a src/WINDOWS directory that contains project files for Microsoft Visual Studio 2005, which should also work with later versions of VS. That directory contains a README.txt file which provides instructions for building LAMMPS from source code using Visual Studio that are hopefully easy to follow for Windows and VS users.

Four VS project options are provided. The first includes the default packages (MANYBODY, MOLECULE, and KSPACE). The second includes all standard packages (except GPU, MEAM, and REAX which are not yet included because they require NVIDIA or Fortran compilation). The third includes all standard packages (with the exceptions) and some user packages. The included user packages are USER−EFF, USER−CG−CMM, and USER−REAXC. The fourth project includes the USER−AWPMD package.

2.3 Making LAMMPS with optional packages

This section has the following sub−sections:

- Package basics
- Including/excluding packages
- Packages that require extra LAMMPS libraries
- Additional Makefile settings for extra libraries

Package basics:

The source code for LAMMPS is structured as a set of core files which are always included, plus optional packages. Packages are groups of files that enable a specific set of features. For example, force fields for molecular systems or granular systems are in packages. You can see the list of all packages by typing "make package".

The current list of standard packages is as follows:

asphere	aspherical particles and force fields
class2	class 2 force fields
colloid	colloidal particle force fields
dipole	point dipole particles and force fields
dsmc	Direct Simulation Monte Carlo (DMSC) pair style
gpu	GPU−enabled force field styles
granular	force fields and boundary conditions for granular systems
kspace	long−range Ewald and particle−mesh (PPPM) solvers
manybody	metal, 3−body, bond−order potentials
meam	modified embedded atom method (MEAM) potential
molecule	force fields for molecular systems
opt	optimized versions of a few pair potentials
peri	Peridynamics model and potential
poems	coupled rigid body motion
reax	ReaxFF potential
replica	multi−replica methods
shock	methods for MD simulations of shock loading
srd	stochastic rotation dynamics (SRD)
xtc	dump atom snapshots in XTC format

There are also several user−contributed packages which may be as simple as a single additional file (see the src/USER−MISC directory) or many files grouped together which add a specific functionality to the code.

The difference between a *standard* package versus a *user* package is as follows:

Standard packages are supported by the LAMMPS developers and are written in a syntax and style consistent with the rest of LAMMPS. This means we will answer questions about them, debug and fix them if necessary, and keep them compatible with future changes to LAMMPS.

User packages don't necessarily meet these requirements. If you have problems using a feature provided in a user package, you will likely need to contact the contributor directly to get help. Information on how to submit

additions you make to LAMMPS as a user−contributed package is given in this section of the documentation.

Including/excluding packages:

To use or not use a package you must include or exclude it before building LAMMPS. From the src directory, this is typically as simple as:

```
make yes−colloid
make g++
```

or

```
make no−manybody
make g++
```

Some packages have individual files that depend on other packages being included. LAMMPS checks for this and does the right thing. I.e. individual files are only included if their dependencies are already included. Likewise, if a package is excluded, other files dependent on that package are also excluded.

The reason to exclude packages is if you will never run certain kinds of simulations. For some packages, this will keep you from having to build auxiliary libraries (see below), and will also produce a smaller executable which may run a bit faster.

By default, LAMMPS includes only the "kspace", "manybody", and "molecule" packages.

Packages are included or excluded by typing "make yes−name" or "make no−name", where "name" is the name of the package. You can also type "make yes−standard", "make no−standard", "make yes−user", "make no−user", "make yes−all" or "make no−all" to include/exclude various sets of packages. Type "make package" to see the all of the package−related make options.

IMPORTANT NOTE: Inclusion/exclusion of a package works by simply moving files back and forth between the main src directory and sub−directories with the package name (e.g. src/KSPACE, src/USER−ATC), so that the files are seen or not seen when LAMMPS is built. After you have included or excluded a package, you must re−build LAMMPS.

Additional package−related make options exist to help manage LAMMPS files that exist in both the src directory and in package sub−directories. You do not normally need to use these commands unless you are editing LAMMPS files or have downloaded a patch from the LAMMPS WWW site.

Typing "make package−update" will overwrite src files with files from the package sub−directories if the package has been included. It should be used after a patch is installed, since patches only update the files in the package sub−directory, but not the src files. Typing "make package−overwrite" will overwrite files in the package sub−directories with src files.

Typing "make package−status" will show which packages are currently included. Of those that are included, it will list files that are different in the src directory and package sub−directory. Typing "make package−diff" lists all differences between these files. Again, type "make package" to see all of the package−related make options.

Packages that require extra LAMMPS libraries:

A few of the standard and user packages require additional auxiliary libraries be compiled first. If you get a LAMMPS build error about a missing library, this is likely the reason. The source code for these libraries is included in the LAMMPS distribution under the "lib" directory. Look at the lib/README file for a list of these.

Each lib directly has a README file (e.g. lib/reax/README) with instructions on how to build that library. Typically this is done by typing something like:

```
make -f Makefile.g++
```

in the appropriate directory, e.g. in lib/reax. Some of the libraries do not build this way. Again, see the libary README file for details.

In any event, you will need to use a Makefile that is a match for your system. If one of the provided Makefiles is not appropriate for your system you will need to edit or add one. For example, in the case of Fortran−based libraries, your system must have a Fortran compiler, the settings for which will need to be listed in the Makefile.

When you have built one of these libraries, there are 2 things to check:

(1) The file libname.a should now exist in lib/name. E.g. lib/reax/libreax.a. This is the library file LAMMPS will link against. One exception is the lib/cuda library which produces the file liblammpscuda.a, because there is already a system library libcuda.a.

(2) The file Makefile.lammps should exist in lib/name. E.g. lib/cuda/Makefile.lammps. This file may be auto−generated by the build of the library, or you may need to make a copy of the appropriate provided file (e.g. lib/meam/Makefile.lammps.gfortran). Either way you should insure that the settings in this file are appropriate for your system.

There are typically 3 settings in the Makefile.lammps file (unless some are blank or not needed): a SYSINC, SYSPATH, and SYSLIB setting, specific to this package. These are settings the LAMMPS build will import when compiling the LAMMPS package files (not the library files), and linking to the auxiliary library. They typically list any other system libraries needed to support the package and where to find them. An example is the BLAS and LAPACK libraries needed by the USER−ATC package. Or the system libraries that support calling Fortran from C++, as the MEAM and REAX packages do.

Note that if these settings are not correct for your box, the LAMMPS build will likely fail.

2.4 Building LAMMPS as a library

LAMMPS itself can be built as a library, which can then be called from another application or a scripting language. See this section for more info on coupling LAMMPS to other codes. Building LAMMPS as a library is done by typing

```
make makelib
make -f Makefile.lib foo
```

where foo is the machine name. Note that inclusion or exclusion of any desired optional packages should be done before typing "make makelib". The first "make" command will create a current Makefile.lib with all the file names in your src dir. The 2nd "make" command will use it to build LAMMPS as a library. This requires that Makefile.foo have a library target (lib) and system−specific settings for ARCHIVE and ARFLAGS. See Makefile.linux for an example. The build will create the file liblmp_foo.a which another application can link to.

When used from a C++ program, the library allows one or more LAMMPS objects to be instantiated. All of LAMMPS is wrapped in a LAMMPS_NS namespace; you can safely use any of its classes and methods from within your application code, as needed.

When used from a C or Fortran program or a scripting language, the library has a simple function−style interface, provided in src/library.cpp and src/library.h.

See the sample codes couple/simple/simple.cpp and simple.c as examples of C++ and C codes that invoke LAMMPS thru its library interface. There are other examples as well in the couple directory which are discussed in this section of the manual. See this section of the manual for a description of the Python wrapper provided with LAMMPS that operates through the LAMMPS library interface.

The files src/library.cpp and library.h contain the C−style interface to LAMMPS. See this section of the manual for a description of the interface and how to extend it for your needs.

2.5 Running LAMMPS

By default, LAMMPS runs by reading commands from stdin; e.g. lmp_linux < in.file. This means you first create an input script (e.g. in.file) containing the desired commands. This section describes how input scripts are structured and what commands they contain.

You can test LAMMPS on any of the sample inputs provided in the examples or bench directory. Input scripts are named in.* and sample outputs are named log.*.name.P where name is a machine and P is the number of processors it was run on.

Here is how you might run a standard Lennard−Jones benchmark on a Linux box, using mpirun to launch a parallel job:

```
cd src
make linux
cp lmp_linux ../bench
cd ../bench
mpirun −np 4 lmp_linux <in.lj
```

See this page for timings for this and the other benchmarks on various platforms.

On a Windows box, you can skip making LAMMPS and simply download an executable, as described above, though the pre−packaged executables include only certain packages.

To run a LAMMPS executable on a Windows machine, first decide whether you want to download the non−MPI (serial) or the MPI (parallel) version of the executable. Download and save the version you have chosen.

For the non−MPI version, follow these steps:

- Get a command prompt by going to Start−>Run... , then typing "cmd".
- Move to the directory where you have saved lmp_win_no−mpi.exe (e.g. by typing: cd "Documents").
- At the command prompt, type "lmp_win_no−mpi −in in.lj", replacing in.lj with the name of your LAMMPS input script.

For the MPI version, which allows you to run LAMMPS under Windows on multiple processors, follow these steps:

- Download and install MPICH2 for Windows.
- You'll need to use the mpiexec.exe and smpd.exe files from the MPICH2 package. Put them in same directory (or path) as the LAMMPS Windows executable.
- Get a command prompt by going to Start−>Run... , then typing "cmd".
- Move to the directory where you have saved lmp_win_mpi.exe (e.g. by typing: cd "Documents").
- Then type something like this: "mpiexec −np 4 −localonly lmp_win_mpi −in in.lj", replacing in.lj with the name of your LAMMPS input script.
- Note that you may need to provide smpd with a passphrase −−− it doesn't matter what you type.

- In this mode, output may not immediately show up on the screen, so if your input script takes a long time to execute, you may need to be patient before the output shows up.
- Alternatively, you can still use this executable to run on a single processor by typing something like: "lmp_win_mpi −in in.lj".

The screen output from LAMMPS is described in the next section. As it runs, LAMMPS also writes a log.lammps file with the same information.

Note that this sequence of commands copies the LAMMPS executable (lmp_linux) to the directory with the input files. This may not be necessary, but some versions of MPI reset the working directory to where the executable is, rather than leave it as the directory where you launch mpirun from (if you launch lmp_linux on its own and not under mpirun). If that happens, LAMMPS will look for additional input files and write its output files to the executable directory, rather than your working directory, which is probably not what you want.

If LAMMPS encounters errors in the input script or while running a simulation it will print an ERROR message and stop or a WARNING message and continue. See this section for a discussion of the various kinds of errors LAMMPS can or can't detect, a list of all ERROR and WARNING messages, and what to do about them.

LAMMPS can run a problem on any number of processors, including a single processor. In theory you should get identical answers on any number of processors and on any machine. In practice, numerical round−off can cause slight differences and eventual divergence of molecular dynamics phase space trajectories.

LAMMPS can run as large a problem as will fit in the physical memory of one or more processors. If you run out of memory, you must run on more processors or setup a smaller problem.

2.6 Command−line options

At run time, LAMMPS recognizes several optional command−line switches which may be used in any order. Either the full word or a one−or−two letter abbreviation can be used:

- −c or −cuda
- −e or −echo
- −i or −in
- −h or −help
- −l or −log
- −p or −partition
- −pl or −plog
- −ps or −pscreen
- −sc or −screen
- −sf or −suffix
- −v or −var

For example, lmp_ibm might be launched as follows:

```
mpirun −np 16 lmp_ibm −v f tmp.out −l my.log −sc none <in.alloy
mpirun −np 16 lmp_ibm −var f tmp.out −log my.log −screen none  <in.alloy
```

Here are the details on the options:

```
−cuda on/off
```

Explicitly enable or disable CUDA support, as provided by the USER−CUDA package. If LAMMPS is built with this package, as described above in Section 2.3, then by default LAMMPS will run in CUDA mode. If this switch

is set to "off", then it will not, even if it was built with the USER−CUDA package, which means you can run standard LAMMPS or with the GPU package for testing or benchmarking purposes. The only reason to set the switch to "on", is to check if LAMMPS was built with the USER−CUDA package, since an error will be generated if it was not.

```
−echo style
```

Set the style of command echoing. The style can be *none* or *screen* or *log* or *both*. Depending on the style, each command read from the input script will be echoed to the screen and/or logfile. This can be useful to figure out which line of your script is causing an input error. The default value is *log*. The echo style can also be set by using the echo command in the input script itself.

```
−in file
```

Specify a file to use as an input script. This is an optional switch when running LAMMPS in one−partition mode. If it is not specified, LAMMPS reads its input script from stdin − e.g. lmp_linux < in.run. This is a required switch when running LAMMPS in multi−partition mode, since multiple processors cannot all read from stdin.

```
−help
```

Print a list of options compiled into this executable for each LAMMPS style (atom_style, fix, compute, pair_style, bond_style, etc). This can help you know if the command you want to use was included via the appropriate package. LAMMPS will print the info and immediately exit if this switch is used.

```
−log file
```

Specify a log file for LAMMPS to write status information to. In one−partition mode, if the switch is not used, LAMMPS writes to the file log.lammps. If this switch is used, LAMMPS writes to the specified file. In multi−partition mode, if the switch is not used, a log.lammps file is created with hi−level status information. Each partition also writes to a log.lammps.N file where N is the partition ID. If the switch is specified in multi−partition mode, the hi−level logfile is named "file" and each partition also logs information to a file.N. For both one−partition and multi−partition mode, if the specified file is "none", then no log files are created. Using a log command in the input script will override this setting. Option −plog will override the name of the partition log files file.N.

```
−partition 8x2 4 5 ...
```

Invoke LAMMPS in multi−partition mode. When LAMMPS is run on P processors and this switch is not used, LAMMPS runs in one partition, i.e. all P processors run a single simulation. If this switch is used, the P processors are split into separate partitions and each partition runs its own simulation. The arguments to the switch specify the number of processors in each partition. Arguments of the form MxN mean M partitions, each with N processors. Arguments of the form N mean a single partition with N processors. The sum of processors in all partitions must equal P. Thus the command "−partition 8x2 4 5" has 10 partitions and runs on a total of 25 processors.

Note that with MPI installed on a machine (e.g. your desktop), you can run on more (virtual) processors than you have physical processors. This can be useful for running multi−replica simulations, on one or a few processors.

The input script specifies what simulation is run on which partition; see the variable and next commands. This howto section gives examples of how to use these commands in this way. Simulations running on different partitions can also communicate with each other; see the temper command.

```
−plog file
```

Specify the base name for the partition log files, so partition N writes log information to file.N. If file is none, then no partition log files are created. This overrides the filename specified in the −log command−line option. This option is useful when working with large numbers of partitions, allowing the partition log files to be suppressed (−plog none) or placed in a sub−directory (−plog replica_files/log.lammps) If this option is not used the log file for partition N is log.lammps.N or whatever is specified by the −log command−line option.

```
−pscreen file
```

Specify the base name for the partition screen file, so partition N writes screen information to file.N. If file is none, then no partition screen files are created. This overrides the filename specified in the −screen command−line option. This option is useful when working with large numbers of partitions, allowing the partition screen files to be suppressed (−pscreen none) or placed in a sub−directory (−pscreen replica_files/screen) If this option is not used the screen file for partition N is screen.N or whatever is specified by the −screen command−line option.

```
−screen file
```

Specify a file for LAMMPS to write its screen information to. In one−partition mode, if the switch is not used, LAMMPS writes to the screen. If this switch is used, LAMMPS writes to the specified file instead and you will see no screen output. In multi−partition mode, if the switch is not used, hi−level status information is written to the screen. Each partition also writes to a screen.N file where N is the partition ID. If the switch is specified in multi−partition mode, the hi−level screen dump is named "file" and each partition also writes screen information to a file.N. For both one−partition and multi−partition mode, if the specified file is "none", then no screen output is performed. Option −pscreen will override the name of the partition screen files file.N.

```
−suffix style
```

Use variants of various styles if they exist. The specified style can be *opt* or *gpu* or *cuda*. These refer to optional packages that LAMMPS can be built with, as described above in Section 2.3. The "opt" style corrsponds to the OPT package, the "gpu" style to the GPU package, and the "cuda" style to the USER−CUDA package.

As an example, all of the packages provide a pair_style lj/cut variant, with style names lj/cut/opt or lj/cut/gpu or lj/cut/cuda. A variant styles can be specified explicitly in your input script, e.g. pair_style lj/cut/gpu. If the −suffix switch is used, you do not need to modify your input script. The specified suffix (opt,gpu,cuda) is automatically appended whenever your input script command creates a new atom, pair, fix, compute, or run style. atom, pair, fix, compute, or integrate style. If the variant version does not exist, the standard version is created.

The suffix command can also set a suffix and it can also turn off/on any suffix setting made via the command line.

```
−var name value1 value2 ...
```

Specify a variable that will be defined for substitution purposes when the input script is read. "Name" is the variable name which can be a single character (referenced as $x in the input script) or a full string (referenced as ${abc}). An index−style variable will be created and populated with the subsequent values, e.g. a set of filenames. Using this command−line option is equivalent to putting the line "variable name index value1 value2 ..." at the beginning of the input script. Defining an index variable as a command−line argument overrides any setting for the same index variable in the input script, since index variables cannot be re−defined. See the variable command for more info on defining index and other kinds of variables and this section for more info on using variables in input scripts.

2.7 LAMMPS screen output

As LAMMPS reads an input script, it prints information to both the screen and a log file about significant actions it takes to setup a simulation. When the simulation is ready to begin, LAMMPS performs various initializations and prints the amount of memory (in MBytes per processor) that the simulation requires. It also prints details of the initial thermodynamic state of the system. During the run itself, thermodynamic information is printed periodically, every few timesteps. When the run concludes, LAMMPS prints the final thermodynamic state and a total run time for the simulation. It then appends statistics about the CPU time and storage requirements for the simulation. An example set of statistics is shown here:

```
Loop time of 49.002 on 2 procs for 2004 atoms

Pair   time (%) = 35.0495 (71.5267)
Bond   time (%) = 0.092046 (0.187841)
Kspce  time (%) = 6.42073 (13.103)
Neigh  time (%) = 2.73485 (5.5811)
Comm   time (%) = 1.50291 (3.06703)
Outpt  time (%) = 0.013799 (0.0281601)
Other  time (%) = 2.13669 (4.36041)

Nlocal:    1002 ave, 1015 max, 989 min
Histogram: 1 0 0 0 0 0 0 0 0 1
Nghost:    8720 ave, 8724 max, 8716 min
Histogram: 1 0 0 0 0 0 0 0 0 1
Neighs:    354141 ave, 361422 max, 346860 min
Histogram: 1 0 0 0 0 0 0 0 0 1

Total # of neighbors = 708282
Ave neighs/atom = 353.434
Ave special neighs/atom = 2.34032
Number of reneighborings = 42
Dangerous reneighborings = 2
```

The first section gives the breakdown of the CPU run time (in seconds) into major categories. The second section lists the number of owned atoms (Nlocal), ghost atoms (Nghost), and pair−wise neighbors stored per processor. The max and min values give the spread of these values across processors with a 10−bin histogram showing the distribution. The total number of histogram counts is equal to the number of processors.

The last section gives aggregate statistics for pair−wise neighbors and special neighbors that LAMMPS keeps track of (see the special_bonds command). The number of times neighbor lists were rebuilt during the run is given as well as the number of potentially "dangerous" rebuilds. If atom movement triggered neighbor list rebuilding (see the neigh_modify command), then dangerous reneighborings are those that were triggered on the first timestep atom movement was checked for. If this count is non−zero you may wish to reduce the delay factor to insure no force interactions are missed by atoms moving beyond the neighbor skin distance before a rebuild takes place.

If an energy minimization was performed via the minimize command, additional information is printed, e.g.

```
Minimization stats:
  E initial, next-to-last, final = -0.895962 -2.94193 -2.94342
  Gradient 2-norm init/final= 1920.78 20.9992
  Gradient inf-norm init/final= 304.283 9.61216
  Iterations = 36
  Force evaluations = 177
```

The first line lists the initial and final energy, as well as the energy on the next−to−last iteration. The next 2 lines give a measure of the gradient of the energy (force on all atoms). The 2−norm is the "length" of this force vector;

the inf−norm is the largest component. The last 2 lines are statistics on how many iterations and force−evaluations the minimizer required. Multiple force evaluations are typically done at each iteration to perform a 1d line minimization in the search direction.

If a kspace_style long−range Coulombics solve was performed during the run (PPPM, Ewald), then additional information is printed, e.g.

```
FFT time (% of Kspce) = 0.200313 (8.34477)
FFT Gflps 3d 1d−only = 2.31074 9.19989
```

The first line gives the time spent doing 3d FFTs (4 per timestep) and the fraction it represents of the total KSpace time (listed above). Each 3d FFT requires computation (3 sets of 1d FFTs) and communication (transposes). The total flops performed is 5Nlog_2(N), where N is the number of points in the 3d grid. The FFTs are timed with and without the communication and a Gflop rate is computed. The 3d rate is with communication; the 1d rate is without (just the 1d FFTs). Thus you can estimate what fraction of your FFT time was spent in communication, roughly 75% in the example above.

2.8 Tips for users of previous LAMMPS versions

The current C++ began with a complete rewrite of LAMMPS 2001, which was written in F90. Features of earlier versions of LAMMPS are listed in this section. The F90 and F77 versions (2001 and 99) are also freely distributed as open−source codes; check the LAMMPS WWW Site for distribution information if you prefer those versions. The 99 and 2001 versions are no longer under active development; they do not have all the features of C++ LAMMPS.

If you are a previous user of LAMMPS 2001, these are the most significant changes you will notice in C++ LAMMPS:

(1) The names and arguments of many input script commands have changed. All commands are now a single word (e.g. read_data instead of read data).

(2) All the functionality of LAMMPS 2001 is included in C++ LAMMPS, but you may need to specify the relevant commands in different ways.

(3) The format of the data file can be streamlined for some problems. See the read_data command for details. The data file section "Nonbond Coeff" has been renamed to "Pair Coeff" in C++ LAMMPS.

(4) Binary restart files written by LAMMPS 2001 cannot be read by C++ LAMMPS with a read_restart command. This is because they were output by F90 which writes in a different binary format than C or C++ writes or reads. Use the *restart2data* tool provided with LAMMPS 2001 to convert the 2001 restart file to a text data file. Then edit the data file as necessary before using the C++ LAMMPS read_data command to read it in.

(5) There are numerous small numerical changes in C++ LAMMPS that mean you will not get identical answers when comparing to a 2001 run. However, your initial thermodynamic energy and MD trajectory should be close if you have setup the problem for both codes the same.

3. Commands

This section describes how a LAMMPS input script is formatted and what commands are used to define a LAMMPS simulation.

3.1 LAMMPS input script
3.2 Parsing rules
3.3 Input script structure
3.4 Commands listed by category
3.5 Commands listed alphabetically

3.1 LAMMPS input script

LAMMPS executes by reading commands from a input script (text file), one line at a time. When the input script ends, LAMMPS exits. Each command causes LAMMPS to take some action. It may set an internal variable, read in a file, or run a simulation. Most commands have default settings, which means you only need to use the command if you wish to change the default.

In many cases, the ordering of commands in an input script is not important. However the following rules apply:

(1) LAMMPS does not read your entire input script and then perform a simulation with all the settings. Rather, the input script is read one line at a time and each command takes effect when it is read. Thus this sequence of commands:

```
timestep 0.5
run      100
run      100
```

does something different than this sequence:

```
run      100
timestep 0.5
run      100
```

In the first case, the specified timestep (0.5 fmsec) is used for two simulations of 100 timesteps each. In the 2nd case, the default timestep (1.0 fmsec) is used for the 1st 100 step simulation and a 0.5 fmsec timestep is used for the 2nd one.

(2) Some commands are only valid when they follow other commands. For example you cannot set the temperature of a group of atoms until atoms have been defined and a group command is used to define which atoms belong to the group.

(3) Sometimes command B will use values that can be set by command A. This means command A must precede command B in the input script if it is to have the desired effect. For example, the read_data command initializes the system by setting up the simulation box and assigning atoms to processors. If default values are not desired, the processors and boundary commands need to be used before read_data to tell LAMMPS how to map processors to the simulation box.

Many input script errors are detected by LAMMPS and an ERROR or WARNING message is printed. This section gives more information on what errors mean. The documentation for each command lists restrictions on

how the command can be used.

3.2 Parsing rules

Each non−blank line in the input script is treated as a command. LAMMPS commands are case sensitive. Command names are lower−case, as are specified command arguments. Upper case letters may be used in file names or user−chosen ID strings.

Here is how each line in the input script is parsed by LAMMPS:

(1) If the last printable character on the line is a ""character (with no surrounding quotes), the command is assumed to continue on the next line. The next line is concatenated to the previous line by removing the ""character and newline. This allows long commands to be continued across two or more lines.

(2) All characters from the first "#" character onward are treated as comment and discarded. See an exception in (6). Note that a comment after a trailing ""character will prevent the command from continuing on the next line. Also note that for multi−line commands a single leading "#" will comment out the entire command.

(3) The line is searched repeatedly for $ characters, which indicate variables that are replaced with a text string. See an exception in (6). If the $ is followed by curly brackets, then the variable name is the text inside the curly brackets. If no curly brackets follow the $, then the variable name is the single character immediately following the $. Thus ${myTemp} and $x refer to variable names "myTemp" and "x". See the variable command for details of how strings are assigned to variables and how they are substituted for in input script commands.

(4) The line is broken into "words" separated by whitespace (tabs, spaces). Note that words can thus contain letters, digits, underscores, or punctuation characters.

(5) The first word is the command name. All successive words in the line are arguments.

(6) If you want text with spaces to be treated as a single argument, it can be enclosed in either double or single quotes. E.g.

```
print "Volume = $v"
print 'Volume = $v'
```

The quotes are removed when the single argument is stored internally. See the dump modify format or if commands for examples. A "#" or "$" character that is between quotes will not be treated as a comment indicator in (2) or substituted for as a variable in (3).

IMPORTANT NOTE: If the argument is itself a command that requires a quoted argument (e.g. using a print command as part of an if or run every command), then the double and single quotes can be nested in the usual manner. See the doc pages for those commands for examples. Only one of level of nesting is allowed, but that should be sufficient for most use cases.

3.3 Input script structure

This section describes the structure of a typical LAMMPS input script. The "examples" directory in the LAMMPS distribution contains many sample input scripts; the corresponding problems are discussed in this section, and animated on the LAMMPS WWW Site.

A LAMMPS input script typically has 4 parts:

1. Initialization
2. Atom definition
3. Settings
4. Run a simulation

The last 2 parts can be repeated as many times as desired. I.e. run a simulation, change some settings, run some more, etc. Each of the 4 parts is now described in more detail. Remember that almost all the commands need only be used if a non−default value is desired.

(1) Initialization

Set parameters that need to be defined before atoms are created or read−in from a file.

The relevant commands are units, dimension, newton, processors, boundary, atom_style, atom_modify.

If force−field parameters appear in the files that will be read, these commands tell LAMMPS what kinds of force fields are being used: pair_style, bond_style, angle_style, dihedral_style, improper_style.

(2) Atom definition

There are 3 ways to define atoms in LAMMPS. Read them in from a data or restart file via the read_data or read_restart commands. These files can contain molecular topology information. Or create atoms on a lattice (with no molecular topology), using these commands: lattice, region, create_box, create_atoms. The entire set of atoms can be duplicated to make a larger simulation using the replicate command.

(3) Settings

Once atoms and molecular topology are defined, a variety of settings can be specified: force field coefficients, simulation parameters, output options, etc.

Force field coefficients are set by these commands (they can also be set in the read−in files): pair_coeff, bond_coeff, angle_coeff, dihedral_coeff, improper_coeff, kspace_style, dielectric, special_bonds.

Various simulation parameters are set by these commands: neighbor, neigh_modify, group, timestep, reset_timestep, run_style, min_style, min_modify.

Fixes impose a variety of boundary conditions, time integration, and diagnostic options. The fix command comes in many flavors.

Various computations can be specified for execution during a simulation using the compute, compute_modify, and variable commands.

Output options are set by the thermo, dump, and restart commands.

(4) Run a simulation

A molecular dynamics simulation is run using the run command. Energy minimization (molecular statics) is performed using the minimize command. A parallel tempering (replica−exchange) simulation can be run using the temper command.

3.4 Commands listed by category

This section lists all LAMMPS commands, grouped by category. The next section lists the same commands alphabetically. Note that some style options for some commands are part of specific LAMMPS packages, which means they cannot be used unless the package was included when LAMMPS was built. Not all packages are included in a default LAMMPS build. These dependencies are listed as Restrictions in the command's documentation.

Initialization:

atom_modify, atom_style, boundary, dimension, newton, processors, units

Atom definition:

create_atoms, create_box, lattice, read_data, read_restart, region, replicate

Force fields:

angle_coeff, angle_style, bond_coeff, bond_style, dielectric, dihedral_coeff, dihedral_style, improper_coeff, improper_style, kspace_modify, kspace_style, pair_coeff, pair_modify, pair_style, pair_write, special_bonds

Settings:

communicate, group, mass, min_modify, min_style, neigh_modify, neighbor, reset_timestep, run_style, set, timestep, velocity

Fixes:

fix, fix_modify, unfix

Computes:

compute, compute_modify, uncompute

Output:

dump, dump image, dump_modify, restart, thermo, thermo_modify, thermo_style, undump, write_restart

Actions:

delete_atoms, delete_bonds, displace_atoms, displace_box, minimize, neb prd, run, temper

Miscellaneous:

clear, echo, if, include, jump, label, log, next, print, shell, variable

---

3.5 Individual commands

This section lists all LAMMPS commands alphabetically, with a separate listing below of styles within certain commands. The previous section lists the same commands, grouped by category. Note that some style options for some commands are part of specific LAMMPS packages, which means they cannot be used unless the package was included when LAMMPS was built. Not all packages are included in a default LAMMPS build. These

dependencies are listed as Restrictions in the command's documentation.

angle_coeff	angle_style	atom_modify	atom_style	bond_coeff	bond_style
boundary	change_box	clear	communicate	compute	compute_modify
create_atoms	create_box	delete_atoms	delete_bonds	dielectric	dihedral_coeff
dihedral_style	dimension	displace_atoms	displace_box	dump	dump image
dump_modify	echo	fix	fix_modify	group	if
improper_coeff	improper_style	include	jump	kspace_modify	kspace_style
label	lattice	log	mass	minimize	min_modify
min_style	neb	neigh_modify	neighbor	newton	next
package	pair_coeff	pair_modify	pair_style	pair_write	prd
print	processors	read_data	read_restart	region	replicate
reset_timestep	restart	run	run_style	set	shell
special_bonds	suffix	tad	temper	thermo	thermo_modify
thermo_style	timestep	uncompute	undump	unfix	units
variable	velocity	write_restart			

Fix styles

See the fix command for one−line descriptions of each style or click on the style itself for a full description:

adapt	addforce	aveforce	ave/atom	ave/correlate	ave/histo	ave/spatial	ave/time
bond/break	bond/create	bond/swap	box/relax	deform	deposit	drag	dt/reset
efield	enforce2d	evaporate	external	freeze	gcmc	gravity	heat
indent	langevin	lineforce	momentum	move	msst	neb	nph
nph/asphere	nph/sphere	npt	npt/asphere	npt/sphere	nve	nve/asphere	nve/limit
nve/noforce	nve/sphere	nvt	nvt/asphere	nvt/sllod	nvt/sphere	orient/fcc	planeforce
poems	pour	press/berendsen	print	qeq/comb	reax/bonds	recenter	restrain
rigid	rigid/nve	rigid/nvt	setforce	shake	spring	spring/rg	spring/self
srd	store/force	store/state	temp/berendsen	temp/rescale	thermal/conductivity	tmd	ttm
viscosity	viscous	wall/colloid	wall/gran	wall/harmonic	wall/lj126	wall/lj93	wall/reflect
wall/region	wall/srd						

These are fix styles contributed by users, which can be used if LAMMPS is built with the appropriate package.

atc	imd	langevin/eff	nph/eff	npt/eff	nve/eff
nvt/eff	nvt/sllod/eff	qeq/reax	smd	temp/rescale/eff	

These are accelerated fix styles, which can be used if LAMMPS is built with the appropriate accelerated package.

freeze/cuda	addforce/cuda	addtorque	aveforce/cuda	enforce2d/cuda	gravity/cuda
npt/cuda	nve/cuda	nvt/cuda	setforce/cuda	shake/cuda	temp/berendsen/cuda
temp/rescale/cuda	temp/rescale/limit/cuda	viscous/cuda			

Compute styles

See the compute command for one−line descriptions of each style or click on the style itself for a full description:

angle/local	atom/molecule	bond/local	centro/atom	cluster/atom	cna/atom
com	com/molecule	coord/atom	damage/atom	dihedral/local	displace/atom
erotate/asphere	erotate/sphere	event/displace	group/group	gyration	gyration/molecule
heat/flux	improper/local	ke	ke/atom	msd	msd/molecule
pair	pair/local	pe	pe/atom	pressure	property/atom
property/local	property/molecule	rdf	reduce	reduce/region	slice
stress/atom	temp	temp/asphere	temp/com	temp/deform	temp/partial
temp/profile	temp/ramp	temp/region	temp/sphere	ti	

These are compute styles contributed by users, which can be used if LAMMPS is built with the appropriate package.

ackland/atom	ke/eff	ke/atom/eff	temp/eff	temp/deform/eff	temp/region/eff
temp/rotate					

These are accelerated compute styles, which can be used if LAMMPS is built with the appropriate accelerated package.

pe/cuda	pressure/cuda	temp/cuda	temp/partial/cuda

Pair_style potentials

See the pair_style command for an overview of pair potentials. Click on the style itself for a full description:

none	hybrid	hybrid/overlay	adp
airebo	born	born/coul/long	buck
buck/coul/cut	buck/coul/long	colloid	comb
coul/cut	coul/debye	coul/long	dipole/cut
dpd	dpd/tstat	dsmc	eam
eam/alloy	eam/fs	eim	gauss
gayberne	gran/hertz/history	gran/hooke	gran/hooke/history
hbond/dreiding/lj	hbond/dreiding/morse	lj/charmm/coul/charmm	lj/charmm/coul/charmm/implicit
lj/charmm/coul/long	lj/class2	lj/class2/coul/cut	lj/class2/coul/long
lj/cut	lj/cut/coul/cut	lj/cut/coul/debye	lj/cut/coul/long
lj/cut/coul/long/tip4p	lj/expand	lj/gromacs	lj/gromacs/coul/gromacs
lj/smooth	lj96/cut	lubricate	meam
morse	peri/lps	peri/pmb	reax
rebo	resquared	soft	sw
table	tersoff	tersoff/zbl	yukawa
yukawa/colloid			

These are pair styles contributed by users, which can be used if LAMMPS is built with the appropriate package.

awpmd/cut	buck/coul	cg/cmm	cg/cmm/coul/cut
cg/cmm/coul/long	dipole/sf	eam/cd	eff/cut

lj/coul	lj/sf	reax/c

These are accelerated pair styles, which can be used if LAMMPS is built with the appropriate accelerated package.

born/coul/long/cuda	buck/coul/cut/cuda	buck/coul/long/cuda	buck/cuda
cg/cmm/coul/cut/cuda	cg/cmm/coul/debye/cuda	cg/cmm/coul/long/cuda	cg/cmm/coul/long/gpu
cg/cmm/cuda	cg/cmm/gpu	coul/long/gpu	eam/alloy/cuda
eam/alloy/opt	eam/cuda	eam/fs/cuda	eam/fs/opt
eam/opt	gayberne/gpu	gran/hooke/cuda	lj/charmm/coul/charmm/cuda
lj/charmm/coul/charmm/implicit/cuda	lj/charmm/coul/long/cuda	lj/charmm/coul/long/gpu	lj/charmm/coul/long/opt
lj/class2/coul/cut/cuda	lj/class2/coul/long/cuda	lj/class2/coul/long/gpu	lj/class2/cuda
lj/class2/gpu	lj/cut/coul/cut/cuda	lj/cut/coul/cut/gpu	lj/cut/coul/debye/cuda
lj/cut/coul/long/cuda	lj/cut/coul/long/gpu	lj/cut/cuda	lj/cut/experimental/cuda
lj/cut/gpu	lj/cut/opt	lj/expand/cuda	lj/expand/gpu
lj/gromacs/coul/gromacs/cuda	lj/gromacs/cuda	lj/smooth/cuda	lj96/cut/cuda
lj96/cut/gpu	morse/cuda	morse/gpu	morse/opt
resquared/gpu			

Bond_style potentials

See the bond_style command for an overview of bond potentials. Click on the style itself for a full description:

none	hybrid	class2	fene
fene/expand	harmonic	morse	nonlinear
quartic	table		

These are bond styles contributed by users, which can be used if LAMMPS is built with the appropriate package.

harmonic/shift	harmonic/shift/cut

Angle_style potentials

See the angle_style command for an overview of angle potentials. Click on the style itself for a full description:

none	hybrid	charmm	class2
cosine	cosine/delta	cosine/periodic	cosine/squared
harmonic	table		

These are angle styles contributed by users, which can be used if LAMMPS is built with the appropriate package.

cg/cmm	cosine/shift	cosine/shift/exp

Dihedral_style potentials

See the dihedral_style command for an overview of dihedral potentials. Click on the style itself for a full description:

none	hybrid	charmm	class2
harmonic	helix	multi/harmonic	opls

These are dihedral styles contributed by users, which can be used if LAMMPS is built with the appropriate package.

cosine/shift/exp

Improper_style potentials

See the improper_style command for an overview of improper potentials. Click on the style itself for a full description:

none	hybrid	class2	cvff
harmonic	umbrella		

Kspace solvers

See the kspace_style command for an overview of Kspace solvers. Click on the style itself for a full description:

ewald	pppm	pppm/cg	pppm/tip4p

These are Kspace solvers contributed by users, which can be used if LAMMPS is built with the appropriate package.

ewald/n

These are accelerated Kspace solvers, which can be used if LAMMPS is built with the appropriate accelerated package.

pppm/cuda	pppm/gpu

5. Using accelerated CPU and GPU styles

Accelerated versions of various pair_style, fixes, computes, and other commands have been added to LAMMPS, which will typically run faster than the standard non−accelerated versions, if you have the appropriate hardware on your system.

The accelerated styles have the same name as the standard styles, except that a suffix is appended. Otherwise, the syntax for the command is identical, their functionality is the same, and the numerical results it produces should also be identical, except for precision and round−off issues.

For example, all of these variants of the basic Lennard−Jones pair style exist in LAMMPS:

- pair_style lj/cut
- pair_style lj/cut/opt
- pair_style lj/cut/omp
- pair_style lj/cut/gpu
- pair_style lj/cut/cuda

Assuming you have built LAMMPS with the appropriate package, these styles can be invoked by specifying them explicitly in your input script. Or you can use the −suffix command−line switch to invoke the accelerated versions automatically, without changing your input script. The suffix command allows you to set a suffix explicitly and to turn off/on the comand−line switch setting, both from within your input script.

Styles with an "opt" suffix are part of the OPT package and typically speed−up the pairwise calculations of your simulation by 5−25%.

Styles with an "omp" suffix are part of the USER−OMP package and allow a pair−style to be run in threaded mode using OpenMP. This can be useful on nodes with high−core counts when using less MPI processes than cores is advantageous, e.g. when running with PPPM so that FFTs are run on fewer MPI processors.

Styles with a "gpu" or "cuda" suffix are part of the GPU or USER−CUDA packages, and can be run on NVIDIA GPUs associated with your CPUs. The speed−up due to GPU usage depends on a variety of factors, as discussed below.

To see what styles are currently available in each of the accelerated packages, see this section of the manual. A list of accelerated styles is included in the pair, fix, compute, and kspace sections.

The following sections explain:

- what hardware and software the accelerated styles require
- how to build LAMMPS with the accelerated packages in place
- what changes (if any) are needed in your input scripts
- guidelines for best performance
- speed−ups you can expect

The final section compares and contrasts the GPU and USER−CUDA packages, since they are both designed to use NVIDIA GPU hardware.

5.1 OPT package
5.2 USER−OMP package

5.3 GPU package
5.4 USER−CUDA package
5.5 Comparison of GPU and USER−CUDA packages

5.1 OPT package

The OPT package was developed by James Fischer (High Performance Technologies), David Richie, and Vincent Natoli (Stone Ridge Technologies). It contains a handful of pair styles whose compute() methods were rewritten in C++ templated form to reduce the overhead due to if tests and other conditional code.

The procedure for building LAMMPS with the OPT package is simple. It is the same as for any other package which has no additional library dependencies:

```
make yes−opt
make machine
```

If your input script uses one of the OPT pair styles, you can run it as follows:

```
lmp_machine −sf opt <in.script
mpirun −np 4 lmp_machine −sf opt  <in.script
```

You should see a reduction in the "Pair time" printed out at the end of the run. On most machines and problems, this will typically be a 5 to 20% savings.

5.2 USER−OMP package

This section will be written when the USER−OMP package is released in main LAMMPS.

5.3 GPU package

The GPU package was developed by Mike Brown at ORNL. It provides GPU versions of several pair styles and for long−range Coulombics via the PPPM command. It has the following features:

- The package is designed to exploit common GPU hardware configurations where one or more GPUs are coupled with many cores of a multi−core CPUs, e.g. within a node of a parallel machine.
- Atom−based data (e.g. coordinates, forces) moves back−and−forth between the CPU(s) and GPU every timestep.
- Neighbor lists can be constructed on the CPU or on the GPU
- The charge assignement and force interpolation portions of PPPM can be run on the GPU. The FFT portion, which requires MPI communication between processors, runs on the CPU.
- Asynchronous force computations can be performed simultaneously on the CPU(s) and GPU.
- LAMMPS−specific code is in the GPU package. It makes calls to a generic GPU library in the lib/gpu directory. This library provides NVIDIA support as well as more general OpenCL support, so that the same functionality can eventually be supported on a variety of GPU hardware.

Hardware and software requirements:

To use this package, you currently need to have specific NVIDIA hardware and install specific NVIDIA CUDA software on your system:

- Check if you have an NVIDIA card: cat /proc/driver/nvidia/cards/0
- Go to http://www.nvidia.com/object/cuda_get.html
- Install a driver and toolkit appropriate for your system (SDK is not necessary)
- Follow the instructions in lammps/lib/gpu/README to build the library (see below)
- Run lammps/lib/gpu/nvc_get_devices to list supported devices and properties

Building LAMMPS with the GPU package:

As with other packages that include a separately compiled library, you need to first build the GPU library, before building LAMMPS itself. General instructions for doing this are in this section of the manual. For this package, do the following, using a Makefile in lib/gpu appropriate for your system:

```
cd lammps/lib/gpu
make -f Makefile.linux
(see further instructions in lammps/lib/gpu/README)
```

If you are successful, you will produce the file lib/libgpu.a.

Now you are ready to build LAMMPS with the GPU package installed:

```
cd lammps/src
make yes-gpu
make machine
```

Note that the lo-level Makefile (e.g. src/MAKE/Makefile.linux) has these settings: gpu_SYSINC, gpu_SYSLIB, gpu_SYSPATH. These need to be set appropriately to include the paths and settings for the CUDA system software on your machine. See src/MAKE/Makefile.g++ for an example.

GPU configuration

When using GPUs, you are restricted to one physical GPU per LAMMPS process, which is an MPI process running on a single core or processor. Multiple MPI processes (CPU cores) can share a single GPU, and in many cases it will be more efficient to run this way.

Input script requirements:

Additional input script requirements to run pair or PPPM styles with a *gpu* suffix are as follows:

- To invoke specific styles from the GPU package, you can either append "gpu" to the style name (e.g. pair_style lj/cut/gpu), or use the -suffix command-line switch, or use the suffix command.
- The newton pair setting must be *off*.
- The package gpu command must be used near the beginning of your script to control the GPU selection and initialization settings. It also has an option to enable asynchronous splitting of force computations between the CPUs and GPUs.

As an example, if you have two GPUs per node and 8 CPU cores per node, and would like to run on 4 nodes (32 cores) with dynamic balancing of force calculation across CPU and GPU cores, you could specify

```
package gpu force/neigh 0 1 -1
```

In this case, all CPU cores and GPU devices on the nodes would be utilized. Each GPU device would be shared by 4 CPU cores. The CPU cores would perform force calculations for some fraction of the particles at the same time the GPUs performed force calculation for the other particles.

Timing output:

As described by the package gpu command, GPU accelerated pair styles can perform computations asynchronously with CPU computations. The "Pair" time reported by LAMMPS will be the maximum of the time required to complete the CPU pair style computations and the time required to complete the GPU pair style computations. Any time spent for GPU−enabled pair styles for computations that run simultaneously with bond, angle, dihedral, improper, and long−range calculations will not be included in the "Pair" time.

When the *mode* setting for the package gpu command is force/neigh, the time for neighbor list calculations on the GPU will be added into the "Pair" time, not the "Neigh" time. An additional breakdown of the times required for various tasks on the GPU (data copy, neighbor calculations, force computations, etc) are output only with the LAMMPS screen output (not in the log file) at the end of each run. These timings represent total time spent on the GPU for each routine, regardless of asynchronous CPU calculations.

Performance tips:

Generally speaking, for best performance, you should use multiple CPUs per GPU, as provided my most multi−core CPU/GPU configurations.

Because of the large number of cores within each GPU device, it may be more efficient to run on fewer processes per GPU when the number of particles per MPI process is small (100's of particles); this can be necessary to keep the GPU cores busy.

See the lammps/lib/gpu/README file for instructions on how to build the GPU library for single, mixed, or double precision. The latter requires that your GPU card support double precision.

5.4 USER−CUDA package

The USER−CUDA package was developed by Christian Trott at U Technology Ilmenau in Germany. It provides NVIDIA GPU versions of many pair styles, many fixes, a few computes, and for long−range Coulombics via the PPPM command. It has the following features:

- The package is designed to allow an entire LAMMPS calculation, for many timesteps, to run entirely on the GPU (except for inter−processor MPI communication), so that atom−based data (e.g. coordinates, forces) do not have to move back−and−forth between the CPU and GPU.
- Data will stay on the GPU until a timestep where a non−GPU−ized fix or compute is invoked. Whenever a non−GPU operation occurs (fix, compute, output), data automatically moves back to the CPU as needed. This may incur a performance penalty, but should otherwise work transparently.
- Neighbor lists for GPU−ized pair styles are constructed on the GPU.
- The package only supports use of a single CPU (core) with each GPU.

Hardware and software requirements:

To use this package, you need to have specific NVIDIA hardware and install specific NVIDIA CUDA software on your system.

Your NVIDIA GPU needs to support Compute Capability 1.3. This list may help you to find out the Compute Capability of your card:

http://en.wikipedia.org/wiki/Comparison_of_Nvidia_graphics_processing_units

Install the Nvidia Cuda Toolkit in version 3.2 or higher and the corresponding GPU drivers. The Nvidia Cuda SDK is not required for LAMMPSCUDA but we recommend it be installed. You can then make sure that its sample projects can be compiled without problems.

Building LAMMPS with the USER−CUDA package:

As with other packages that include a separately compiled library, you need to first build the USER−CUDA library, before building LAMMPS itself. General instructions for doing this are in this section of the manual. For this package, do the following, using settings in the lib/cuda Makefiles appropriate for your system:

- Go to the lammps/lib/cuda directory
- If your *CUDA* toolkit is not installed in the default system directoy */usr/local/cuda* edit the file *lib/cuda/Makefile.common* accordingly.
- Type "make OPTIONS", where *OPTIONS* are one or more of the following options. The settings will be written to the *lib/cuda/Makefile.defaults* and used in the next step.

```
precision=N to set the precision level
  N = 1 for single precision (default)
  N = 2 for double precision
  N = 3 for positions in double precision
  N = 4 for positions and velocities in double precision
arch=M to set GPU compute capability
  M = 20 for CC2.0 (GF100/110, e.g. C2050,GTX580,GTX470) (default)
  M = 21 for CC2.1 (GF104/114,  e.g. GTX560, GTX460, GTX450)
  M = 13 for CC1.3 (GF200, e.g. C1060, GTX285)
prec_timer=0/1 to use hi−precision timers
  0 = do not use them (default)
  1 = use these timers
  this is usually only useful for Mac machines
dbg=0/1 to activate debug mode
  0 = no debug mode (default)
  1 = yes debug mode
  this is only useful for developers
cufft=1 to determine usage of CUDA FFT library
  0 = no CUFFT support (default)
  in the future other CUDA−enabled FFT libraries might be supported
```

- Type "make" to build the library. If you are successful, you will produce the file lib/libcuda.a.

Now you are ready to build LAMMPS with the USER−CUDA package installed:

```
cd lammps/src
make yes−user−cuda
make machine
```

Note that the LAMMPS build references the lib/cuda/Makefile.common file to extract setting specific CUDA settings. So it is important that you have first built the cuda library (in lib/cuda) using settings appropriate to your system.

Input script requirements:

Additional input script requirements to run styles with a *cuda* suffix are as follows:

- To invoke specific styles from the USER−CUDA package, you can either append "cuda" to the style name (e.g. pair_style lj/cut/cuda), or use the −suffix command−line switch, or use the suffix command. One exception is that the kspace_style pppm/cuda command has to be requested explicitly.
- To use the USER−CUDA package with its default settings, no additional command is needed in your input script. This is because when LAMMPS starts up, it detects if it has been built with the

USER−CUDA package. See the −cuda command−line switch for more details.
- To change settings for the USER−CUDA package at run−time, the package cuda command can be used near the beginning of your input script. See the package command doc page for details.

Performance tips:

The USER−CUDA package offers more speed−up relative to CPU performance when the number of atoms per GPU is large, e.g. on the order of tens or hundreds of 1000s.

As noted above, this package will continue to run a simulation entirely on the GPU(s) (except for inter−processor MPI communication), for multiple timesteps, until a CPU calculation is required, either by a fix or compute that is non−GPU−ized, or until output is performed (thermo or dump snapshot or restart file). The less often this occurs, the faster your simulation will run.

5.5 Comparison of GPU and USER−CUDA packages

Both the GPU and USER−CUDA packages accelerate a LAMMPS calculation using NVIDIA hardware, but they do it in different ways.

As a consequence, for a particular simulation on specific hardware, one package may be faster than the other. We give guidelines below, but the best way to determine which package is faster for your input script is to try both of them on your machine. See the benchmarking section below for examples where this has been done.

Guidelines for using each package optimally:

- The GPU package allows you to assign multiple CPUs (cores) to a single GPU (a common configuration for "hybrid" nodes that contain multicore CPU(s) and GPU(s)) and works effectively in this mode. The USER−CUDA package does not allow this; you can only use one CPU per GPU.
- The GPU package moves per−atom data (coordinates, forces) back−and−forth between the CPU and GPU every timestep. The USER−CUDA package only does this on timesteps when a CPU calculation is required (e.g. to invoke a fix or compute that is non−GPU−ized). Hence, if you can formulate your input script to only use GPU−ized fixes and computes, and avoid doing I/O too often (thermo output, dump file snapshots, restart files), then the data transfer cost of the USER−CUDA package can be very low, causing it to run faster than the GPU package.
- The GPU package is often faster than the USER−CUDA package, if the number of atoms per GPU is "small". The crossover point, in terms of atoms/GPU at which the USER−CUDA package becomes faster depends strongly on the pair style. For example, for a simple Lennard Jones system the crossover (in single precision) is often about 50K−100K atoms per GPU. When performing double precision calculations the crossover point can be significantly smaller.
- Both packages compute bonded interactions (bonds, angles, etc) on the CPU. This means a model with bonds will force the USER−CUDA package to transfer per−atom data back−and−forth between the CPU and GPU every timestep. If the GPU package is running with several MPI processes assigned to one GPU, the cost of computing the bonded interactions is spread across more CPUs and hence the GPU package can run faster.
- When using the GPU package with multiple CPUs assigned to one GPU, its performance depends to some extent on high bandwidth between the CPUs and the GPU. Hence its performance is affected if full 16 PCIe lanes are not available for each GPU. In HPC environments this can be the case if S2050/70 servers are used, where two devices generally share one PCIe 2.0 16x slot. Also many multi−GPU mainboards do not provide full 16 lanes to each of the PCIe 2.0 16x slots.

Differences between the two packages:

- The GPU package accelerates only pair force, neighbor list, and PPPM calculations. The USER−CUDA package currently supports a wider range of pair styles and can also accelerate many fix styles and some compute styles, as well as neighbor list and PPPM calculations.
- The GPU package uses more GPU memory than the USER−CUDA package. This is generally not a problem since typical runs are computation−limited rather than memory−limited.

Examples:

The LAMMPS distribution has two directories with sample input scripts for the GPU and USER−CUDA packages.

- lammps/examples/gpu = GPU package files
- lammps/examples/USER/cuda = USER−CUDA package files

These contain input scripts for identical systems, so they can be used to benchmark the performance of both packages on your system.

Benchmark data:

NOTE: We plan to add some benchmark results and plots here for the examples described in the previous section.

Simulations:

1. Lennard Jones

- 256,000 atoms
- 2.5 A cutoff
- 0.844 density

2. Lennard Jones

- 256,000 atoms
- 5.0 A cutoff
- 0.844 density

3. Rhodopsin model

- 256,000 atoms
- 10A cutoff
- Coulomb via PPPM

4. Lihtium−Phosphate

- 295650 atoms
- 15A cutoff
- Coulomb via PPPM

Hardware:

Workstation:

- 2x GTX470

- i7 950@3GHz
- 24Gb DDR3 @ 1066Mhz
- CentOS 5.5
- CUDA 3.2
- Driver 260.19.12

eStella:

- 6 Nodes
- 2xC2050
- 2xQDR Infiniband interconnect(aggregate bandwidth 80GBps)
- Intel X5650 HexCore @ 2.67GHz
- SL 5.5
- CUDA 3.2
- Driver 260.19.26

Keeneland:

- HP SL−390 (Ariston) cluster
- 120 nodes
- 2x Intel Westmere hex−core CPUs
- 3xC2070s
- QDR InfiniBand interconnect

6. How−to discussions

The following sections describe how to use various options within LAMMPS.

6.1 Restarting a simulation
6.2 2d simulations
6.3 CHARMM, AMBER, and DREIDING force fields
6.4 Running multiple simulations from one input script
6.5 Multi−replica simulations
6.6 Granular models
6.7 TIP3P water model
6.8 TIP4P water model
6.9 SPC water model
6.10 Coupling LAMMPS to other codes
6.11 Visualizing LAMMPS snapshots
6.12 Triclinic (non−orthogonal) simulation boxes
6.13 NEMD simulations
6.14 Extended spherical and aspherical particles
6.15 Output from LAMMPS (thermo, dumps, computes, fixes, variables)
6.16 Thermostatting, barostatting and computing temperature
6.17 Walls
6.18 Elastic constants
6.19 Library interface to LAMMPS
6.20 Calculating thermal conductivity
6.21 Calculating viscosity

The example input scripts included in the LAMMPS distribution and highlighted in this section also show how to setup and run various kinds of simulations.

6.1 Restarting a simulation

There are 3 ways to continue a long LAMMPS simulation. Multiple run commands can be used in the same input script. Each run will continue from where the previous run left off. Or binary restart files can be saved to disk using the restart command. At a later time, these binary files can be read via a read_restart command in a new script. Or they can be converted to text data files and read by a read_data command in a new script. This section discusses the *restart2data* tool that is used to perform the conversion.

Here we give examples of 2 scripts that read either a binary restart file or a converted data file and then issue a new run command to continue where the previous run left off. They illustrate what settings must be made in the new script. Details are discussed in the documentation for the read_restart and read_data commands.

Look at the *in.chain* input script provided in the *bench* directory of the LAMMPS distribution to see the original script that these 2 scripts are based on. If that script had the line

```
restart         50 tmp.restart
```

added to it, it would produce 2 binary restart files (tmp.restart.50 and tmp.restart.100) as it ran.

This script could be used to read the 1st restart file and re−run the last 50 timesteps:

```
read_restart    tmp.restart.50

neighbor        0.4 bin
neigh_modify    every 1 delay 1

fix             1 all nve
fix             2 all langevin 1.0 1.0 10.0 904297

timestep        0.012

run             50
```

Note that the following commands do not need to be repeated because their settings are included in the restart file: *units*, *atom_style*, *special_bonds*, *pair_style*, *bond_style*. However these commands do need to be used, since their settings are not in the restart file: *neighbor*, *fix*, *timestep*.

If you actually use this script to perform a restarted run, you will notice that the thermodynamic data match at step 50 (if you also put a "thermo 50" command in the original script), but do not match at step 100. This is because the fix langevin command uses random numbers in a way that does not allow for perfect restarts.

As an alternate approach, the restart file could be converted to a data file using this tool:

```
restart2data tmp.restart.50 tmp.restart.data
```

Then, this script could be used to re−run the last 50 steps:

```
units           lj
atom_style      bond
pair_style      lj/cut 1.12
pair_modify     shift yes
bond_style      fene
special_bonds   0.0 1.0 1.0

read_data       tmp.restart.data

neighbor        0.4 bin
neigh_modify    every 1 delay 1

fix             1 all nve
fix             2 all langevin 1.0 1.0 10.0 904297

timestep        0.012

reset_timestep  50
run             50
```

Note that nearly all the settings specified in the original *in.chain* script must be repeated, except the *pair_coeff* and *bond_coeff* commands since the new data file lists the force field coefficients. Also, the reset_timestep command is used to tell LAMMPS the current timestep. This value is stored in restart files, but not in data files.

6.2 2d simulations

Use the dimension command to specify a 2d simulation.

Make the simulation box periodic in z via the boundary command. This is the default.

If using the create box command to define a simulation box, set the z dimensions narrow, but finite, so that the create_atoms command will tile the 3d simulation box with a single z plane of atoms − e.g.

```
create box 1 −10 10 −10 10 −0.25 0.25
```

If using the read data command to read in a file of atom coordinates, set the "zlo zhi" values to be finite but narrow, similar to the create_box command settings just described. For each atom in the file, assign a z coordinate so it falls inside the z−boundaries of the box − e.g. 0.0.

Use the fix enforce2d command as the last defined fix to insure that the z−components of velocities and forces are zeroed out every timestep. The reason to make it the last fix is so that any forces induced by other fixes will be zeroed out.

Many of the example input scripts included in the LAMMPS distribution are for 2d models.

IMPORTANT NOTE: Some models in LAMMPS treat particles as extended spheres, as opposed to point particles. In 2d, the particles will still be spheres, not disks, meaning their moment of inertia will be the same as in 3d.

6.3 CHARMM, AMBER, and DREIDING force fields

A force field has 2 parts: the formulas that define it and the coefficients used for a particular system. Here we only discuss formulas implemented in LAMMPS that correspond to formulas commonly used in the CHARMM, AMBER, and DREIDING force fields. Setting coefficients is done in the input data file via the read_data command or in the input script with commands like pair_coeff or bond_coeff. See this section for additional tools that can use CHARMM or AMBER to assign force field coefficients and convert their output into LAMMPS input.

See (MacKerell) for a description of the CHARMM force field. See (Cornell) for a description of the AMBER force field.

These style choices compute force field formulas that are consistent with common options in CHARMM or AMBER. See each command's documentation for the formula it computes.

- bond_style harmonic
- angle_style charmm
- dihedral_style charmm
- pair_style lj/charmm/coul/charmm
- pair_style lj/charmm/coul/charmm/implicit
- pair_style lj/charmm/coul/long

- special_bonds charmm
- special_bonds amber

DREIDING is a generic force field developed by the Goddard group at Caltech and is useful for predicting structures and dynamics of organic, biological and main−group inorganic molecules. The philosophy in DREIDING is to use general force constants and geometry parameters based on simple hybridization considerations, rather than individual force constants and geometric parameters that depend on the particular combinations of atoms involved in the bond, angle, or torsion terms. DREIDING has an explicit hydrogen bond term to describe interactions involving a hydrogen atom on very electronegative atoms (N, O, F).

See (Mayo) for a description of the DREIDING force field

These style choices compute force field formulas that are consistent with the DREIDING force field. See each command's documentation for the formula it computes.

- bond_style harmonic
- bond_style morse

- angle_style harmonic
- angle_style cosine
- angle_style cosine/periodic

- dihedral_style charmm
- improper_style umbrella

- pair_style buck
- pair_style buck/coul/cut
- pair_style buck/coul/long
- pair_style lj/cut
- pair_style lj/cut/coul/cut
- pair_style lj/cut/coul/long

- pair_style hbond/dreiding/lj
- pair_style hbond/dreiding/morse

- special_bonds dreiding

6.4 Running multiple simulations from one input script

This can be done in several ways. See the documentation for individual commands for more details on how these examples work.

If "multiple simulations" means continue a previous simulation for more timesteps, then you simply use the run command multiple times. For example, this script

```
units lj
atom_style atomic
read_data data.lj
run 10000
run 10000
run 10000
run 10000
run 10000
```

would run 5 successive simulations of the same system for a total of 50,000 timesteps.

If you wish to run totally different simulations, one after the other, the clear command can be used in between them to re−initialize LAMMPS. For example, this script

```
units lj
atom_style atomic
read_data data.lj
run 10000
clear
```

```
units lj
atom_style atomic
read_data data.lj.new
run 10000
```

would run 2 independent simulations, one after the other.

For large numbers of independent simulations, you can use variables and the next and jump commands to loop over the same input script multiple times with different settings. For example, this script, named in.polymer

```
variable d index run1 run2 run3 run4 run5 run6 run7 run8
shell cd $d
read_data data.polymer
run 10000
shell cd ..
clear
next d
jump in.polymer
```

would run 8 simulations in different directories, using a data.polymer file in each directory. The same concept could be used to run the same system at 8 different temperatures, using a temperature variable and storing the output in different log and dump files, for example

```
variable a loop 8
variable t index 0.8 0.85 0.9 0.95 1.0 1.05 1.1 1.15
log log.$a
read data.polymer
velocity all create $t 352839
fix 1 all nvt $t $t 100.0
dump 1 all atom 1000 dump.$a
run 100000
next t
next a
jump in.polymer
```

All of the above examples work whether you are running on 1 or multiple processors, but assumed you are running LAMMPS on a single partition of processors. LAMMPS can be run on multiple partitions via the "−partition" command−line switch as described in this section of the manual.

In the last 2 examples, if LAMMPS were run on 3 partitions, the same scripts could be used if the "index" and "loop" variables were replaced with *universe*−style variables, as described in the variable command. Also, the "next t" and "next a" commands would need to be replaced with a single "next a t" command. With these modifications, the 8 simulations of each script would run on the 3 partitions one after the other until all were finished. Initially, 3 simulations would be started simultaneously, one on each partition. When one finished, that partition would then start the 4th simulation, and so forth, until all 8 were completed.

6.5 Multi−replica simulations

Several commands in LAMMPS run mutli−replica simulations, meaning that multiple instances (replicas) of your simulation are run simultaneously, with small amounts of data exchanged between replicas periodically.

These are the relevant commands:

- neb for nudged elastic band calculations
- prd for parallel replica dynamics

- tad for temperature accelerated dynamics
- temper for parallel tempering

NEB is a method for finding transition states and barrier energies. PRD and TAD are methods for performing accelerated dynamics to find and perform infrequent events. Parallel tempering or replica exchange runs different replicas at a series of temperature to facilitate rare−event sampling.

These command can only be used if LAMMPS was built with the "replica" package. See the Making LAMMPS section for more info on packages.

In all these cases, you must run with one or more processors per replica. The processors assigned to each replica are determined at run−time by using the −partition command−line switch to launch LAMMPS on multiple partitions, which in this context are the same as replicas. E.g. these commands:

```
mpirun −np 16 lmp_linux −partition 8x2 −in in.temper
mpirun −np 8 lmp_linux −partition 8x1 −in in.neb
```

would each run 8 replicas, on either 16 or 8 processors. Note the use of the −in command−line switch to specify the input script which is required when running in multi−replica mode.

Also note that with MPI installed on a machine (e.g. your desktop), you can run on more (virtual) processors than you have physical processors. Thus the above commands could be run on a single−processor (or few−processor) desktop so that you can run a multi−replica simulation on more replicas than you have physical processors.

6.6 Granular models

Granular system are composed of spherical particles with a diameter, as opposed to point particles. This means they have an angular velocity and torque can be imparted to them to cause them to rotate.

To run a simulation of a granular model, you will want to use the following commands:

- atom_style sphere
- fix nve/sphere
- fix gravity

This compute

- compute erotate/sphere

calculates rotational kinetic energy which can be output with thermodynamic info.

Use one of these 3 pair potentials, which compute forces and torques between interacting pairs of particles:

- pair_style gran/history
- pair_style gran/no_history
- pair_style gran/hertzian

These commands implement fix options specific to granular systems:

- fix freeze
- fix pour
- fix viscous

- fix wall/gran

The fix style *freeze* zeroes both the force and torque of frozen atoms, and should be used for granular system instead of the fix style *setforce*.

For computational efficiency, you can eliminate needless pairwise computations between frozen atoms by using this command:

- neigh_modify exclude

6.7 TIP3P water model

The TIP3P water model as implemented in CHARMM (MacKerell) specifies a 3−site rigid water molecule with charges and Lennard−Jones parameters assigned to each of the 3 atoms. In LAMMPS the fix shake command can be used to hold the two O−H bonds and the H−O−H angle rigid. A bond style of *harmonic* and an angle style of *harmonic* or *charmm* should also be used.

These are the additional parameters (in real units) to set for O and H atoms and the water molecule to run a rigid TIP3P−CHARMM model with a cutoff. The K values can be used if a flexible TIP3P model (without fix shake) is desired. If the LJ epsilon and sigma for HH and OH are set to 0.0, it corresponds to the original 1983 TIP3P model (Jorgensen).

O mass = 15.9994
H mass = 1.008

O charge = −0.834
H charge = 0.417

LJ epsilon of OO = 0.1521
LJ sigma of OO = 3.1507
LJ epsilon of HH = 0.0460
LJ sigma of HH = 0.4000
LJ epsilon of OH = 0.0836
LJ sigma of OH = 1.7753

K of OH bond = 450
r0 of OH bond = 0.9572

K of HOH angle = 55
theta of HOH angle = 104.52

These are the parameters to use for TIP3P with a long−range Coulombic solver (Ewald or PPPM in LAMMPS), see (Price) for details:

O mass = 15.9994
H mass = 1.008

O charge = −0.830
H charge = 0.415

LJ epsilon of OO = 0.102

LJ sigma of OO = 3.188
LJ epsilon, sigma of OH, HH = 0.0

K of OH bond = 450
r0 of OH bond = 0.9572

K of HOH angle = 55
theta of HOH angle = 104.52

Wikipedia also has a nice article on water models.

6.8 TIP4P water model

The four−point TIP4P rigid water model extends the traditional three−point TIP3P model by adding an additional site, usually massless, where the charge associated with the oxygen atom is placed. This site M is located at a fixed distance away from the oxygen along the bisector of the HOH bond angle. A bond style of *harmonic* and an angle style of *harmonic* or *charmm* should also be used.

Currently, only a four−point model for long−range Coulombics is implemented via the LAMMPS pair style lj/cut/coul/long/tip4p. A cutoff version may be added the future. For both models, the bond lengths and bond angles should be held fixed using the fix shake command.

These are the additional parameters (in real units) to set for O and H atoms and the water molecule to run a rigid TIP4P model with a cutoff (Jorgensen). Note that the OM distance is specified in the pair_style command, not as part of the pair coefficients.

O mass = 15.9994
H mass = 1.008

O charge = −1.040
H charge = 0.520

r0 of OH bond = 0.9572
theta of HOH angle = 104.52

OM distance = 0.15

LJ epsilon of O−O = 0.1550
LJ sigma of O−O = 3.1536
LJ epsilon, sigma of OH, HH = 0.0

These are the parameters to use for TIP4P with a long−range Coulombic solver (Ewald or PPPM in LAMMPS):

O mass = 15.9994
H mass = 1.008

O charge = −1.0484
H charge = 0.5242

r0 of OH bond = 0.9572
theta of HOH angle = 104.52

OM distance = 0.1250

LJ epsilon of O−O = 0.16275
LJ sigma of O−O = 3.16435
LJ epsilon, sigma of OH, HH = 0.0

Wikipedia also has a nice article on water models.

6.9 SPC water model

The SPC water model specifies a 3−site rigid water molecule with charges and Lennard−Jones parameters assigned to each of the 3 atoms. In LAMMPS the fix shake command can be used to hold the two O−H bonds and the H−O−H angle rigid. A bond style of *harmonic* and an angle style of *harmonic* or *charmm* should also be used.

These are the additional parameters (in real units) to set for O and H atoms and the water molecule to run a rigid SPC model.

O mass = 15.9994
H mass = 1.008

O charge = −0.820
H charge = 0.410

LJ epsilon of OO = 0.1553
LJ sigma of OO = 3.166
LJ epsilon, sigma of OH, HH = 0.0

r0 of OH bond = 1.0
theta of HOH angle = 109.47

Note that as originally proposed, the SPC model was run with a 9 Angstrom cutoff for both LJ and Coulommbic terms. It can also be used with long−range Coulombics (Ewald or PPPM in LAMMPS), without changing any of the parameters above, though it becomes a different model in that mode of usage.

The SPC/E (extended) water model is the same, except the partial charge assignemnts change:

O charge = −0.8476
H charge = 0.4238

See the (Berendsen) reference for more details on both the SPC and SPC/E models.

Wikipedia also has a nice article on water models.

6.10 Coupling LAMMPS to other codes

LAMMPS is designed to allow it to be coupled to other codes. For example, a quantum mechanics code might compute forces on a subset of atoms and pass those forces to LAMMPS. Or a continuum finite element (FE) simulation might use atom positions as boundary conditions on FE nodal points, compute a FE solution, and return interpolated forces on MD atoms.

LAMMPS can be coupled to other codes in at least 3 ways. Each has advantages and disadvantages, which you'll have to think about in the context of your application.

(1) Define a new fix command that calls the other code. In this scenario, LAMMPS is the driver code. During its timestepping, the fix is invoked, and can make library calls to the other code, which has been linked to LAMMPS as a library. This is the way the POEMS package that performs constrained rigid−body motion on groups of atoms is hooked to LAMMPS. See the fix_poems command for more details. See this section of the documentation for info on how to add a new fix to LAMMPS.

(2) Define a new LAMMPS command that calls the other code. This is conceptually similar to method (1), but in this case LAMMPS and the other code are on a more equal footing. Note that now the other code is not called during the timestepping of a LAMMPS run, but between runs. The LAMMPS input script can be used to alternate LAMMPS runs with calls to the other code, invoked via the new command. The run command facilitates this with its *every* option, which makes it easy to run a few steps, invoke the command, run a few steps, invoke the command, etc.

In this scenario, the other code can be called as a library, as in (1), or it could be a stand−alone code, invoked by a system() call made by the command (assuming your parallel machine allows one or more processors to start up another program). In the latter case the stand−alone code could communicate with LAMMPS thru files that the command writes and reads.

See this section of the documentation for how to add a new command to LAMMPS.

(3) Use LAMMPS as a library called by another code. In this case the other code is the driver and calls LAMMPS as needed. Or a wrapper code could link and call both LAMMPS and another code as libraries. Again, the run command has options that allow it to be invoked with minimal overhead (no setup or clean−up) if you wish to do multiple short runs, driven by another program.

Examples of driver codes that call LAMMPS as a library are included in the "couple" directory of the LAMMPS distribution; see couple/README for more details:

- simple: simple driver programs in C++ and C which invoke LAMMPS as a library
- lammps_quest: coupling of LAMMPS and Quest, to run classical MD with quantum forces calculated by a density functional code
- lammps_spparks: coupling of LAMMPS and SPPARKS, to couple a kinetic Monte Carlo model for grain growth using MD to calculate strain induced across grain boundaries

This section of the documentation describes how to build LAMMPS as a library. Once this is done, you can interface with LAMMPS either via C++, C, Fortran, or Python (or any other language that supports a vanilla C−like interface). For example, from C++ you could create one (or more) "instances" of LAMMPS, pass it an input script to process, or execute individual commands, all by invoking the correct class methods in LAMMPS. From C or Fortran you can make function calls to do the same things. See this section of the manual for a description of the Python wrapper provided with LAMMPS that operates through the LAMMPS library interface.

The files src/library.cpp and library.h contain the C−style interface to LAMMPS. See this section of the manual for a description of the interface and how to extend it for your needs.

Note that the lammps_open() function that creates an instance of LAMMPS takes an MPI communicator as an argument. This means that instance of LAMMPS will run on the set of processors in the communicator. Thus the calling code can run LAMMPS on all or a subset of processors. For example, a wrapper script might decide to alternate between LAMMPS and another code, allowing them both to run on all the processors. Or it might allocate half the processors to LAMMPS and half to the other code and run both codes simultaneously before

syncing them up periodically. Or it might instantiate multiple instances of LAMMPS to perform different calculations.

6.11 Visualizing LAMMPS snapshots

LAMMPS itself does not do visualization, but snapshots from LAMMPS simulations can be visualized (and analyzed) in a variety of ways.

LAMMPS snapshots are created by the dump command which can create files in several formats. The native LAMMPS dump format is a text file (see "dump atom" or "dump custom") which can be visualized by the xmovie program, included with the LAMMPS package. This produces simple, fast 2d projections of 3d systems, and can be useful for rapid debugging of simulation geometry and atom trajectories.

Several programs included with LAMMPS as auxiliary tools can convert native LAMMPS dump files to other formats. See the Section_tools doc page for details. The first is the ch2lmp tool, which contains a lammps2pdb Perl script which converts LAMMPS dump files into PDB files. The second is the lmp2arc tool which converts LAMMPS dump files into Accelrys' Insight MD program files. The third is the lmp2cfg tool which converts LAMMPS dump files into CFG files which can be read into the AtomEye visualizer.

A Python−based toolkit distributed by our group can read native LAMMPS dump files, including custom dump files with additional columns of user−specified atom information, and convert them to various formats or pipe them into visualization software directly. See the Pizza.py WWW site for details. Specifically, Pizza.py can convert LAMMPS dump files into PDB, XYZ, Ensight, and VTK formats. Pizza.py can pipe LAMMPS dump files directly into the Raster3d and RasMol visualization programs. Pizza.py has tools that do interactive 3d OpenGL visualization and one that creates SVG images of dump file snapshots.

LAMMPS can create XYZ files directly (via "dump xyz") which is a simple text−based file format used by many visualization programs including VMD.

LAMMPS can create DCD files directly (via "dump dcd") which can be read by VMD in conjunction with a CHARMM PSF file. Using this form of output avoids the need to convert LAMMPS snapshots to PDB files. See the dump command for more information on DCD files.

LAMMPS can create XTC files directly (via "dump xtc") which is GROMACS file format which can also be read by VMD for visualization. See the dump command for more information on XTC files.

6.12 Triclinic (non−orthogonal) simulation boxes

By default, LAMMPS uses an orthogonal simulation box to encompass the particles. The boundary command sets the boundary conditions of the box (periodic, non−periodic, etc). The orthogonal box has its "origin" at (xlo,ylo,zlo) and is defined by 3 edge vectors starting from the origin given by **a** = (xhi−xlo,0,0); **b** = (0,yhi−ylo,0); **c** = (0,0,zhi−zlo). The 6 parameters (xlo,xhi,ylo,yhi,zlo,zhi) are defined at the time the simluation box is created, e.g. by the create_box or read_data or read_restart commands. Additionally, LAMMPS defines box size parameters lx,ly,lz where lx = xhi−xlo, and similarly in the y and z dimensions. The 6 parameters, as well as lx,ly,lz, can be output via the thermo_style custom command.

LAMMPS also allows simulations to be perfored in non−orthogonal simulation boxes shaped as a parallelepiped with triclinic symmetry. The parallelepiped has its "origin" at (xlo,ylo,zlo) and is defined by 3 edge vectors starting from the origin given by **a** = (xhi−xlo,0,0); **b** = (xy,yhi−ylo,0); **c** = (xz,yz,zhi−zlo). *Xy*,*xz*,*yz* can be 0.0 or positive or negative values and are called "tilt factors" because they are the amount of displacement applied to

faces of an originally orthogonal box to transform it into the parallelepiped. Note that in LAMMPS the triclinic simulation box edge vectors **a**, **b**, and **c** cannot be arbitrary vectors. As indicated, **a** must be aligned with the x axis, **b** must be in the xy plane, and **c** is arbitrary. However, this is not a restriction since it is possible to rotate any set of 3 crystal basis vectors so that they meet this restriction.

The 9 parameters (xlo,xhi,ylo,yhi,zlo,zhi,xy,xz,yz) are defined at the time the simluation box is created. This happens in one of 3 ways. If the create_box command is used with a region of style *prism*, then a triclinic box is setup. See the region command for details. If the read_data command is used to define the simulation box, and the header of the data file contains a line with the "xy xz yz" keyword, then a triclinic box is setup. See the read_data command for details. Finally, if the read_restart command reads a restart file which was written from a simulation using a triclinic box, then a triclinic box will be setup for the restarted simulation.

Note that you can define a triclinic box with all 3 tilt factors = 0.0, so that it is initially orthogonal. This is necessary if the box will become non−orthogonal, e.g. due to the fix npt or fix deform commands. Alternatively, you can use the change_box command to convert a simulation box from orthogonal to triclinic and vice versa.

As with orthogonal boxes, LAMMPS defines triclinic box size parameters lx,ly,lz where lx = xhi−xlo, and similarly in the y and z dimensions. The 9 parameters, as well as lx,ly,lz, can be output via the thermo_style custom command.

To avoid extremely tilted boxes (which would be computationally inefficient), no tilt factor can skew the box more than half the distance of the parallel box length, which is the 1st dimension in the tilt factor (x for xz). For example, if xlo = 2 and xhi = 12, then the x box length is 10 and the xy tilt factor must be between −5 and 5. Similarly, both xz and yz must be between −(xhi−xlo)/2 and +(yhi−ylo)/2. Note that this is not a limitation, since if the maximum tilt factor is 5 (as in this example), then configurations with tilt = ..., −15, −5, 5, 15, 25, ... are geometrically all equivalent.

Triclinic crystal structures are often defined using three lattice constants *a*, *b*, and *c*, and three angles *alpha*, *beta* and *gamma*. Note that in this nomenclature, the a, b, and c lattice constants are the scalar lengths of the edge vectors **a**, **b**, and **c** defined above. The relationship between these 6 quantities (a,b,c,alpha,beta,gamma) and the LAMMPS box sizes (lx,ly,lz) = (xhi−xlo,yhi−ylo,zhi−zlo) and tilt factors (xy,xz,yz) is as follows:

$$
\begin{aligned}
a &= \mathrm{lx} \\
b^2 &= \mathrm{ly}^2 + \mathrm{xy}^2 \\
c^2 &= \mathrm{lz}^2 + \mathrm{xz}^2 + \mathrm{yz}^2 \\
\cos\alpha &= \frac{\mathrm{xy} * \mathrm{xz} + \mathrm{ly} * \mathrm{yz}}{b * c} \\
\cos\beta &= \frac{\mathrm{xz}}{c} \\
\cos\gamma &= \frac{\mathrm{xy}}{b}
\end{aligned}
$$

The inverse relationship can be written as follows:

$$
\begin{aligned}
\mathrm{lx} &= a \\
\mathrm{xy} &= b\cos\gamma \\
\mathrm{xz} &= c\cos\beta \\
\mathrm{ly}^2 &= b^2 - \mathrm{xy}^2 \\
\mathrm{yz} &= \frac{b*c\cos\alpha - \mathrm{xy}*\mathrm{xz}}{\mathrm{ly}} \\
\mathrm{lz}^2 &= c^2 - \mathrm{xz}^2 - \mathrm{yz}^2
\end{aligned}
$$

The values of *a*, *b*, *c* , *alpha*, *beta* , and *gamma* can be printed out or accessed by computes using the thermo_style custom keywords *cella*, *cellb*, *cellc*, *cellalpha*, *cellbeta*, *cellgamma*, respectively.

As discussed on the dump command doc page, when the BOX BOUNDS for a snapshot is written to a dump file for a triclinic box, an orthogonal bounding box which encloses the triclinic simulation box is output, along with the 3 tilt factors (xy, xz, yz) of the triclinic box, formatted as follows:

```
ITEM: BOX BOUNDS xy xz yz
xlo_bound xhi_bound xy
ylo_bound yhi_bound xz
zlo_bound zhi_bound yz
```

This bounding box is convenient for many visualization programs and is calculated from the 9 triclinic box parameters (xlo,xhi,ylo,yhi,zlo,zhi,xy,xz,yz) as follows:

```
xlo_bound = xlo + MIN(0.0,xy,xz,xy+xz)
xhi_bound = xhi + MAX(0.0,xy,xz,xy+xz)
ylo_bound = ylo + MIN(0.0,yz)
yhi_bound = yhi + MAX(0.0,yz)
zlo_bound = zlo
zhi_bound = zhi
```

These formulas can be inverted if you need to convert the bounding box back into the triclinic box parameters, e.g. xlo = xlo_bound − MIN(0.0,xy,xz,xy+xz).

One use of triclinic simulation boxes is to model solid−state crystals with triclinic symmetry. The lattice command can be used with non−orthogonal basis vectors to define a lattice that will tile a triclinic simulation box via the create_atoms command.

A second use is to run Parinello−Rahman dyanamics via the fix npt command, which will adjust the xy, xz, yz tilt factors to compensate for off−diagonal components of the pressure tensor. The analalog for an energy minimization is the fix box/relax command.

A third use is to shear a bulk solid to study the response of the material. The fix deform command can be used for this purpose. It allows dynamic control of the xy, xz, yz tilt factors as a simulation runs. This is discussed in the next section on non−equilibrium MD (NEMD) simulations.

6.13 NEMD simulations

Non−equilibrium molecular dynamics or NEMD simulations are typically used to measure a fluid's rheological properties such as viscosity. In LAMMPS, such simulations can be performed by first setting up a non−orthogonal simulation box (see the preceding Howto section).

A shear strain can be applied to the simulation box at a desired strain rate by using the fix deform command. The fix nvt/sllod command can be used to thermostat the sheared fluid and integrate the SLLOD equations of motion for the system. Fix nvt/sllod uses compute temp/deform to compute a thermal temperature by subtracting out the streaming velocity of the shearing atoms. The velocity profile or other properties of the fluid can be monitored via the fix ave/spatial command.

As discussed in the previous section on non−orthogonal simulation boxes, the amount of tilt or skew that can be applied is limited by LAMMPS for computational efficiency to be 1/2 of the parallel box length. However, fix deform can continuously strain a box by an arbitrary amount. As discussed in the fix deform command, when the tilt value reaches a limit, the box is re−shaped to the opposite limit which is an equivalent tiling of periodic space. The strain rate can then continue to change as before. In a long NEMD simulation these box re−shaping events may occur many times.

In a NEMD simulation, the "remap" option of fix deform should be set to "remap v", since that is what fix nvt/sllod assumes to generate a velocity profile consistent with the applied shear strain rate.

An alternative method for calculating viscosities is provided via the fix viscosity command.

6.14 Extended spherical and aspherical particles

Typical MD models treat atoms or particles as point masses. Sometimes, however, it is desirable to have a model with finite−size particles such as spheres or aspherical ellipsoids. The difference is that such particles have a moment of inertia, rotational energy, and angular momentum. Rotation is induced by torque from interactions with other particles.

LAMMPS has several options for running simulations with these kinds of particles. The following aspects are discussed in turn:

- atom styles
- pair potentials
- time integration
- computes, thermodynamics, and dump output
- rigid bodies composed of extended particles

Atom styles

There are 2 atom styles that allow for definition of finite−size particles: sphere and ellipsoid. The peri atom style also treats particles as having a volume, but that is internal to the pair_style peri potentials. The dipole atom style is most often used in conjunction with finite−size particles.

The sphere style defines particles that are spheriods and each particle can have a unique diameter and mass (or density). These particles store an angular velocity (omega) and can be acted upon by torque. The "set" command can be used to modify the diameter and mass of individual particles, after then are created.

The ellipsoid style defines particles that are ellipsoids and thus can be aspherical. Each particle has a shape, specified by 3 diameters, and mass (or density). These particles store an angular momentum and their orientation (quaternion), and can be acted upon by torque. They do not store an angular velocity (omega), which can be in a different direction than angular momentum, rather they compute it as needed. The "set" command can be used to modify the diameter, orientation, and mass of individual particles, after then are created. It also has a brief explanation of what quaternions are.

The dipole style does not define extended particles, but is often used in conjunction with spherical particles, via a command like

```
atom_style hybrid sphere dipole
```

This is because when dipoles interact with each other, they induce torques, and a particle must be extended (i.e. have a moment of inertia) in order to respond and rotate. See the atom_style dipole command for details. The "set" command can be used to modify the orientation and length of the dipole moment of individual particles, after then are created.

Note that if one of these atom styles is used (or multiple styles via the atom_style hybrid command), not all particles in the system are required to be finite−size or aspherical. For example, if the 3 shape parameters are set to the same value, the particle will be a sphere rather than an ellipsoid. If the 3 shape parameters are all set to 0.0 or if the diameter is set to 0.0, it will be a point particle. If the length of the dipole moment is set to zero, the particle will not have a point dipole associated with it. The pair styles used to compute pairwise interactions will typically compute the correct interaction in these simplified (cheaper) cases. Pair_style hybrid can be used to insure the correct interactions are computed for the appropriate style of interactions. Likewise, using groups to partition particles (ellipsoids versus spheres versus point particles) will allow you to use the appropriate time integrators and temperature computations for each class of particles. See the doc pages for various commands for details.

Also note that for 2d simulations, finite−size spheres and ellipsoids are still treated as 3d particles, rather than as circular disks or ellipses. This means they have the same moment of inertia for a 3d extended object. When their temperature is coomputed, the correct degrees of freedom are used for rotation in a 2d versus 3d system.

Pair potentials

When a system with extended particles is defined, the particles will only rotate and experience torque if the force field computes such interactions. These are the various pair styles that generate torque:

- pair_style gran/history
- pair_style gran/hertzian
- pair_style gran/no_history
- pair_style dipole/cut
- pair_style gayberne
- pair_style resquared
- pair_style lubricate

The granular pair styles are used with spherical particles. The dipole pair style is used with atom_style dipole, which could be applied to spherical or ellipsoidal particles. The GayBerne and REsquared potentials require ellipsoidal particles, though they will also work if the 3 shape parameters are the same (a sphere). The lubrication potential works with spherical particles.

Time integration

There are 3 fixes that perform time integration on extended spherical particles, meaning the integrators update the rotational orientation and angular velocity or angular momentum of the particles:

- fix nve/sphere
- fix nvt/sphere
- fix npt/sphere

Likewise, there are 3 fixes that perform time integration on ellipsoids as extended aspherical particles:

- fix nve/asphere
- fix nvt/asphere
- fix npt/asphere

The advantage of these fixes is that those which thermostat the particles include the rotational degrees of freedom in the temperature calculation and thermostatting. Other thermostats can be used with fix nve/sphere or fix nve/asphere, such as fix langevin or fix temp/berendsen, but those thermostats only operate on the translational kinetic energy of the extended particles.

Note that for mixtures of point and extended particles, you should only use these integration fixes on groups which contain extended particles.

Computes, thermodynamics, and dump output

There are 4 computes that calculate the temperature or rotational energy of extended spherical or aspherical particles (ellipsoids):

- compute temp/sphere
- compute temp/asphere
- compute erotate/sphere
- compute erotate/asphere

These include rotational degrees of freedom in their computation. If you wish the thermodynamic output of temperature or pressure to use one of these computes (e.g. for a system entirely composed of extended particles), then the compute can be defined and the thermo_modify command used. Note that by default thermodynamic quantities will be calculated with a temperature that only includes translational degrees of freedom. See the thermo_style command for details.

The dump custom command can output various attributes of extended particles, including the dipole moment (mu), the angular velocity (omega), the angular momentum (angmom), the quaternion (quat), and the torque (tq) on the particle.

Rigid bodies composed of extended particles

The fix rigid command treats a collection of particles as a rigid body, computes its inertia tensor, sums the total force and torque on the rigid body each timestep due to forces on its constituent particles, and integrates the motion of the rigid body.

If any of the constituent particles of a rigid body are extended particles (spheres or ellipsoids), then their contribution to the inertia tensor of the body is different than if they were point particles. This means the rotational dynamics of the rigid body will be different. Thus a model of a dimer is different if the dimer consists of two point masses versus two extended sphereoids, even if the two particles have the same mass. Extended particles that experience torque due to their interaction with other particles will also impart that torque to a rigid body they are part of.

See the "fix rigid" command for example of complex rigid−body models it is possible to define in LAMMPS.

Note that the fix shake command can also be used to treat 2, 3, or 4 particles as a rigid body, but it always assumes the particles are point masses.

6.15 Output from LAMMPS (thermo, dumps, computes, fixes, variables)

There are four basic kinds of LAMMPS output:

- Thermodynamic output, which is a list of quantities printed every few timesteps to the screen and logfile.
- Dump files, which contain snapshots of atoms and various per−atom values and are written at a specified frequency.
- Certain fixes can output user−specified quantities to files: fix ave/time for time averaging, fix ave/spatial for spatial averaging, and fix print for single−line output of variables. Fix print can also output to the screen.
- Restart files.

A simulation prints one set of thermodynamic output and (optionally) restart files. It can generate any number of dump files and fix output files, depending on what dump and fix commands you specify.

As discussed below, LAMMPS gives you a variety of ways to determine what quantities are computed and printed when the thermodynamics, dump, or fix commands listed above perform output. Throughout this discussion, note that users can also add their own computes and fixes to LAMMPS which can then generate values that can then be output with these commands.

The following sub−sections discuss different LAMMPS command related to output and the kind of data they operate on and produce:

- Global/per−atom/local data
- Scalar/vector/array data
- Thermodynamic output
- Dump file output
- Fixes that write output files
- Computes that process output quantities
- Fixes that process output quantities
- Computes that generate values to output
- Fixes that generate values to output
- Variables that generate values to output
- Summary table of output options and data flow between commands

Global/per−atom/local data

Various output−related commands work with three different styles of data: global, per−atom, or local. A global datum is one or more system−wide values, e.g. the temperature of the system. A per−atom datum is one or more values per atom, e.g. the kinetic energy of each atom. Local datums are calculated by each processor based on the atoms it owns, but there may be zero or more per atom, e.g. a list of bond distances.

Scalar/vector/array data

Global, per−atom, and local datums can each come in three kinds: a single scalar value, a vector of values, or a 2d array of values. The doc page for a "compute" or "fix" or "variable" that generates data will specify both the style and kind of data it produces, e.g. a per−atom vector.

When a quantity is accessed, as in many of the output commands discussed below, it can be referenced via the following bracket notation, where ID in this case is the ID of a compute. The leading "c_" would be replaced by "f_" for a fix, or "v_" for a variable:

c_ID	entire scalar, vector, or array

c_ID[I]	one element of vector, one column of array
c_ID[I][J]	one element of array

In other words, using one bracket reduces the dimension of the data once (vector −> scalar, array −> vector). Using two brackets reduces the dimension twice (array −> scalar). Thus a command that uses scalar values as input can typically also process elements of a vector or array.

Thermodynamic output

The frequency and format of thermodynamic output is set by the thermo, thermo_style, and thermo_modify commands. The thermo_style command also specifies what values are calculated and written out. Pre−defined keywords can be specified (e.g. press, etotal, etc). Three additional kinds of keywords can also be specified (c_ID, f_ID, v_name), where a compute or fix or variable provides the value to be output. In each case, the compute, fix, or variable must generate global values for input to the thermo_style custom command.

Dump file output

Dump file output is specified by the dump and dump_modify commands. There are several pre−defined formats (dump atom, dump xtc, etc).

There is also a dump custom format where the user specifies what values are output with each atom. Pre−defined atom attributes can be specified (id, x, fx, etc). Three additional kinds of keywords can also be specified (c_ID, f_ID, v_name), where a compute or fix or variable provides the values to be output. In each case, the compute, fix, or variable must generate per−atom values for input to the dump custom command.

There is also a dump local format where the user specifies what local values to output. A pre−defined index keyword can be specified to enumuerate the local values. Two additional kinds of keywords can also be specified (c_ID, f_ID), where a compute or fix or variable provides the values to be output. In each case, the compute or fix must generate local values for input to the dump local command.

Fixes that write output files

Sevarl fixes take various quantities as input and can write output files: fix ave/time, fix ave/spatial, fix ave/histo, fix ave/correlate, and fix print.

The fix ave/time command enables direct output to a file and/or time−averaging of global scalars or vectors. The user specifies one or more quantities as input. These can be global compute values, global fix values, or variables of any style except the atom style which produces per−atom values. Since a variable can refer to keywords used by the thermo_style custom command (like temp or press) and individual per−atom values, a wide variety of quantities can be time averaged and/or output in this way. If the inputs are one or more scalar values, then the fix generate a global scalar or vector of output. If the inputs are one or more vector values, then the fix generates a global vector or array of output. The time−averaged output of this fix can also be used as input to other output commands.

The fix ave/spatial command enables direct output to a file of spatial−averaged per−atom quantities like those output in dump files, within 1d layers of the simulation box. The per−atom quantities can be atom density (mass or number) or atom attributes such as position, velocity, force. They can also be per−atom quantities calculated by a compute, by a fix, or by an atom−style variable. The spatial−averaged output of this fix can also be used as input to other output commands.

The fix ave/histo command enables direct output to a file of histogrammed quantities, which can be global or per−atom or local quantities. The histogram output of this fix can also be used as input to other output commands.

The fix ave/correlate command enables direct output to a file of time−correlated quantities, which can be global scalars. The correlation matrix output of this fix can also be used as input to other output commands.

The fix print command can generate a line of output written to the screen and log file or to a separate file, periodically during a running simulation. The line can contain one or more variable values for any style variable except the atom style). As explained above, variables themselves can contain references to global values generated by thermodynamic keywords, computes, fixes, or other variables, or to per−atom values for a specific atom. Thus the fix print command is a means to output a wide variety of quantities separate from normal thermodynamic or dump file output.

Computes that process output quantities

The compute reduce and compute reduce/region commands take one or more per−atom or local vector quantities as inputs and "reduce" them (sum, min, max, ave) to scalar quantities. These are produced as output values which can be used as input to other output commands.

The compute slice command take one or more global vector or array quantities as inputs and extracts a subset of their values to create a new vector or array. These are produced as output values which can be used as input to other output commands.

The compute property/atom command takes a list of one or more pre−defined atom attributes (id, x, fx, etc) and stores the values in a per−atom vector or array. These are produced as output values which can be used as input to other output commands. The list of atom attributes is the same as for the dump custom command.

The compute property/local command takes a list of one or more pre−defined local attributes (bond info, angle info, etc) and stores the values in a local vector or array. These are produced as output values which can be used as input to other output commands.

The compute atom/molecule command takes a list of one or more per−atom quantities (from a compute, fix, per−atom variable) and sums the quantities on a per−molecule basis. It produces a global vector or array as output values which can be used as input to other output commands.

Fixes that process output quantities

The fix ave/atom command performs time−averaging of per−atom vectors. The per−atom quantities can be atom attributes such as position, velocity, force. They can also be per−atom quantities calculated by a compute, by a fix, or by an atom−style variable. The time−averaged per−atom output of this fix can be used as input to other output commands.

The fix store/state command can archive one or more per−atom attributes at a particular time, so that the old values can be used in a future calculation or output. The list of atom attributes is the same as for the dump custom command, including per−atom quantities calculated by a compute, by a fix, or by an atom−style variable. The output of this fix can be used as input to other output commands.

Computes that generate values to output

Every compute in LAMMPS produces either global or per−atom or local values. The values can be scalars or vectors or arrays of data. These values can be output using the other commands described in this section. The doc page for each compute command describes what it produces. Computes that produce per−atom or local values have the word "atom" or "local" in their style name. Computes without the word "atom" or "local" produce global values.

Fixes that generate values to output

Some fixes in LAMMPS produces either global or per−atom or local values which can be accessed by other commands. The values can be scalars or vectors or arrays of data. These values can be output using the other commands described in this section. The doc page for each fix command tells whether it produces any output quantities and describes them.

Variables that generate values to output

Every variables defined in an input script generates either a global scalar value or a per−atom vector (only atom−style variables) when it is accessed. The formulas used to define equal− and atom−style variables can contain references to the thermodynamic keywords and to global and per−atom data generated by computes, fixes, and other variables. The values generated by variables can be output using the other commands described in this section.

Summary table of output options and data flow between commands

This table summarizes the various commands that can be used for generating output from LAMMPS. Each command produces output data of some kind and/or writes data to a file. Most of the commands can take data from other commands as input. Thus you can link many of these commands together in pipeline form, where data produced by one command is used as input to another command and eventually written to the screen or to a file. Note that to hook two commands together the output and input data types must match, e.g. global/per−atom/local data and scalar/vector/array data.

Also note that, as described above, when a command takes a scalar as input, that could be an element of a vector or array. Likewise a vector input could be a column of an array.

Command	Input	Output
thermo_style custom	global scalars	screen, log file
dump custom	per−atom vectors	dump file
dump local	local vectors	dump file
fix print	global scalar from variable	screen, file
print	global scalar from variable	screen
computes	N/A	global/per−atom/local scalar/vector/array
fixes	N/A	global/per−atom/local scalar/vector/array
variables	global scalars, per−atom vectors	global scalar, per−atom vector
compute reduce	per−atom/local vectors	global scalar/vector
compute slice	global vectors/arrays	global vector/array
compute property/atom	per−atom vectors	per−atom vector/array
compute property/local	local vectors	local vector/array
compute atom/molecule	per−atom vectors	global vector/array
fix ave/atom	per−atom vectors	per−atom vector/array
fix ave/time	global scalars/vectors	global scalar/vector/array, file
fix ave/spatial	per−atom vectors	global array, file
fix ave/histo	global/per−atom/local scalars and vectors	global array, file
fix ave/correlate	global scalars	global array, file
fix store/state	per−atom vectors	per−atom vector/array

6.16 Thermostatting, barostatting, and computing temperature

Thermostatting means controlling the temperature of particles in an MD simulation. Barostatting means controlling the pressure. Since the pressure includes a kinetic component due to particle velocities, both these operations require calculation of the temperature. Typically a target temperature (T) and/or pressure (P) is specified by the user, and the thermostat or barostat attempts to equilibrate the system to the requested T and/or P.

Temperature is computed as kinetic energy divided by some number of degrees of freedom (and the Boltzmann constant). Since kinetic energy is a function of particle velocity, there is often a need to distinguish between a particle's advection velocity (due to some aggregate motiion of particles) and its thermal velocity. The sum of the two is the particle's total velocity, but the latter is often what is wanted to compute a temperature.

LAMMPS has several options for computing temperatures, any of which can be used in thermostatting and barostatting. These compute commands calculate temperature, and the compute pressure command calculates pressure.

- compute temp
- compute temp/sphere
- compute temp/asphere
- compute temp/com
- compute temp/deform
- compute temp/partial
- compute temp/profile
- compute temp/ramp
- compute temp/region

All but the first 3 calculate velocity biases (i.e. advection velocities) that are removed when computing the thermal temperature. Compute temp/sphere and compute temp/asphere compute kinetic energy for extended particles that includes rotational degrees of freedom. They both allow, as an extra argument, which is another temperature compute that subtracts a velocity bias. This allows the translational velocity of extended spherical or aspherical particles to be adjusted in prescribed ways.

Thermostatting in LAMMPS is performed by fixes, or in one case by a pair style. Four thermostatting fixes are currently available: Nose−Hoover (nvt), Berendsen, Langevin, and direct rescaling (temp/rescale). Dissipative particle dynamics (DPD) thermostatting can be invoked via the *dpd/tstat* pair style:

- fix nvt
- fix nvt/sphere
- fix nvt/asphere
- fix nvt/sllod
- fix temp/berendsen
- fix langevin
- fix temp/rescale
- pair_style dpd/tstat

Fix nvt only thermostats the translational velocity of particles. Fix nvt/sllod also does this, except that it subtracts out a velocity bias due to a deforming box and integrates the SLLOD equations of motion. See the NEMD simulations section of this page for further details. Fix nvt/sphere and fix nvt/asphere thermostat not only translation velocities but also rotational velocities for spherical and aspherical particles.

DPD thermostatting alters pairwise interactions in a manner analagous to the per−particle thermostatting of fix langevin.

Any of the thermostatting fixes can use temperature computes that remove bias for two purposes: (a) computing the current temperature to compare to the requested target temperature, and (b) adjusting only the thermal temperature component of the particle's velocities. See the doc pages for the individual fixes and for the fix_modify command for instructions on how to assign a temperature compute to a thermostatting fix. For example, you can apply a thermostat to only the x and z components of velocity by using it in conjunction with compute temp/partial.

IMPORTANT NOTE: Only the nvt fixes perform time integration, meaning they update the velocities and positions of particles due to forces and velocities respectively. The other thermostat fixes only adjust velocities; they do NOT perform time integration updates. Thus they should be used in conjunction with a constant NVE integration fix such as these:

- fix nve
- fix nve/sphere
- fix nve/asphere

Barostatting in LAMMPS is also performed by fixes. Two barosttating methods are currently available: Nose−Hoover (npt and nph) and Berendsen:

- fix npt
- fix npt/sphere
- fix npt/asphere
- fix nph
- fix press/berendsen

The fix npt commands include a Nose−Hoover thermostat and barostat. Fix nph is just a Nose/Hoover barostat; it does no thermostatting. Both fix nph and fix press/bernendsen can be used in conjunction with any of the thermostatting fixes.

As with the thermostats, fix npt and fix nph only use translational motion of the particles in computing T and P and performing thermo/barostatting. Fix npt/sphere and fix npt/asphere thermo/barostat using not only translation velocities but also rotational velocities for spherical and aspherical particles.

All of the barostatting fixes use the compute pressure compute to calculate a current pressure. By default, this compute is created with a simple compute temp (see the last argument of the compute pressure command), which is used to calculated the kinetic componenet of the pressure. The barostatting fixes can also use temperature computes that remove bias for the purpose of computing the kinetic componenet which contributes to the current pressure. See the doc pages for the individual fixes and for the fix_modify command for instructions on how to assign a temperature or pressure compute to a barostatting fix.

IMPORTANT NOTE: As with the thermostats, the Nose/Hoover methods (fix npt and fix nph) perform time integration. Fix press/berendsen does NOT, so it should be used with one of the constant NVE fixes or with one of the NVT fixes.

Finally, thermodynamic output, which can be setup via the thermo_style command, often includes temperature and pressure values. As explained on the doc page for the thermo_style command, the default T and P are setup by the thermo command itself. They are NOT the ones associated with any thermostatting or barostatting fix you have defined or with any compute that calculates a temperature or pressure. Thus if you want to view these values of T and P, you need to specify them explicitly via a thermo_style custom command. Or you can use the thermo_modify command to re−define what temperature or pressure compute is used for default thermodynamic output.

6.17 Walls

Walls in an MD simulation are typically used to bound particle motion, i.e. to serve as a boundary condition.

Walls in LAMMPS can be of rough (made of particles) or idealized surfaces. Ideal walls can be smooth, generating forces only in the normal direction, or frictional, generating forces also in the tangential direction.

Rough walls, built of particles, can be created in various ways. The particles themselves can be generated like any other particle, via the lattice and create_atoms commands, or read in via the read_data command.

Their motion can be constrained by many different commands, so that they do not move at all, move together as a group at constant velocity or in response to a net force acting on them, move in a prescribed fashion (e.g. rotate around a point), etc. Note that if a time integration fix like fix nve or fix nvt is not used with the group that contains wall particles, their positions and velocities will not be updated.

- fix aveforce − set force on particles to average value, so they move together
- fix setforce − set force on particles to a value, e.g. 0.0
- fix freeze − freeze particles for use as granular walls
- fix nve/noforce − advect particles by their velocity, but without force
- fix move − prescribe motion of particles by a linear velocity, oscillation, rotation, variable

The fix move command offers the most generality, since the motion of individual particles can be specified with variable formula which depends on time and/or the particle position.

For rough walls, it may be useful to turn off pairwise interactions between wall particles via the neigh_modify exclude command.

Rough walls can also be created by specifying frozen particles that do not move and do not interact with mobile particles, and then tethering other particles to the fixed particles, via a bond. The bonded particles do interact with other mobile particles.

Idealized walls can be specified via several fix commands. Fix wall/gran creates frictional walls for use with granular particles; all the other commands create smooth walls.

- fix wall/reflect − reflective flat walls
- fix wall/lj93 − flat walls, with Lennard−Jones 9/3 potential
- fix wall/lj126 − flat walls, with Lennard−Jones 12/6 potential
- fix wall/colloid − flat walls, with pair_style colloid potential
- fix wall/harmonic − flat walls, with repulsive harmonic spring potential
- fix wall/region − use region surface as wall
- fix wall/gran − flat or curved walls with pair_style granular potential

The *lj93*, *lj126*, *colloid*, and *harmonic* styles all allow the flat walls to move with a constant velocity, or oscillate in time. The fix wall/region command offers the most generality, since the region surface is treated as a wall, and the geometry of the region can be a simple primitive volume (e.g. a sphere, or cube, or plane), or a complex volume made from the union and intersection of primitive volumes. Regions can also specify a volume "interior" or "exterior" to the specified primitive shape or *union* or *intersection*. Regions can also be "dynamic" meaning they move with constant velocity, oscillate, or rotate.

The only frictional idealized walls currently in LAMMPS are flat or curved surfaces specified by the fix wall/gran command. At some point we plan to allow regoin surfaces to be used as frictional walls, as well as triangulated surfaces.

6.18 Elastic constants

Elastic constants characterize the stiffness of a material. The formal definition is provided by the linear relation that holds between the stress and strain tensors in the limit of infinitesimal deformation. In tensor notation, this is expressed as s_ij = C_ijkl * e_kl, where the repeated indices imply summation. s_ij are the elements of the symmetric stress tensor. e_kl are the elements of the symmetric strain tensor. C_ijkl are the elements of the fourth rank tensor of elastic constants. In three dimensions, this tensor has 3^4=81 elements. Using Voigt notation, the tensor can be written as a 6x6 matrix, where C_ij is now the derivative of s_i w.r.t. e_j. Because s_i is itself a derivative w.r.t. e_i, it follows that C_ij is also symmetric, with at most 7*6/2 = 21 distinct elements.

At zero temperature, it is easy to estimate these derivatives by deforming the cell in one of the six directions using the command displace_box and measuring the change in the stress tensor. A general−purpose script that does this is given in the examples/elastic directory described in this section.

Calculating elastic constants at finite temperature is more challenging, because it is necessary to run a simulation that perfoms time averages of differential properties. One way to do this is to measure the change in average stress tensor in an NVT simulations when the cell volume undergoes a finite deformation. In order to balance the systematic and statistical errors in this method, the magnitude of the deformation must be chosen judiciously, and care must be taken to fully equilibrate the deformed cell before sampling the stress tensor. Another approach is to sample the triclinic cell fluctuations that occur in an NPT simulation. This method can also be slow to converge and requires careful post−processing (Shinoda)

6.19 Library interface to LAMMPS

As described in this section, LAMMPS can be built as a library, so that it can be called by another code, used in a coupled manner with other codes, or driven through a Python interface.

All of these methodologies use a C−style interface to LAMMPS that is provided in the files src/library.cpp and src/library.h. The functions therein have a C−style argument list, but contain C++ code you could write yourself in a C++ application that was invoking LAMMPS directly. The C++ code in the functions illustrates how to invoke internal LAMMPS operations. Note that LAMMPS classes are defined within a LAMMPS namespace (LAMMPS_NS) if you use them from another C++ application.

Library.cpp contains these 4 functions:

```
void lammps_open(int, char **, MPI_Comm, void **);
void lammps_close(void *);
void lammps_file(void *, char *);
char *lammps_command(void *, char *);
```

The lammps_open() function is used to initialize LAMMPS, passing in a list of strings as if they were command−line arguments when LAMMPS is run in stand−alone mode from the command line, and a MPI communicator for LAMMPS to run under. It returns a ptr to the LAMMPS object that is created, and which is used in subsequent library calls. The lammps_open() function can be called multiple times, to create multiple instances of LAMMPS.

LAMMPS will run on the set of processors in the communicator. This means the calling code can run LAMMPS on all or a subset of processors. For example, a wrapper script might decide to alternate between LAMMPS and another code, allowing them both to run on all the processors. Or it might allocate half the processors to LAMMPS and half to the other code and run both codes simultaneously before syncing them up periodically. Or

it might instantiate multiple instances of LAMMPS to perform different calculations.

The lammps_close() function is used to shut down an instance of LAMMPS and free all its memory.

The lammps_file() and lammps_command() functions are used to pass a file or string to LAMMPS as if it were an input script or single command in an input script. Thus the calling code can read or generate a series of LAMMPS commands one line at a time and pass it thru the library interface to setup a problem and then run it, interleaving the lammps_command() calls with other calls to extract information from LAMMPS, perform its own operations, or call another code's library.

Other useful functions are also included in library.cpp. For example:

```
void *lammps_extract_global(void *, char *)
void *lammps_extract_atom(void *, char *)
void *lammps_extract_compute(void *, char *, int, int)
void *lammps_extract_fix(void *, char *, int, int, int, int)
void *lammps_extract_variable(void *, char *, char *)
int lammps_get_natoms(void *)
void lammps_get_coords(void *, double *)
void lammps_put_coords(void *, double *)
```

These can extract various global or per−atom quantities from LAMMPS as well as values calculated by a compute, fix, or variable. The "get" and "put" operations can retrieve and reset atom coordinates. See the library.cpp file and its associated header file library.h for details.

The key idea of the library interface is that you can write any functions you wish to define how your code talks to LAMMPS and add them to src/library.cpp and src/library.h, as well as to the Python interface. The routines you add can access or change any LAMMPS data you wish. The couple and python directories have example C++ and C and Python codes which show how a driver code can link to LAMMPS as a library, run LAMMPS on a subset of processors, grab data from LAMMPS, change it, and put it back into LAMMPS.

6.20 Calculating thermal conductivity

The thermal conductivity kappa of a material can be measured in at least 3 ways using various options in LAMMPS. (See this section of the manual for an analogous discussion for viscosity). The thermal conducitivity tensor kappa is a measure of the propensity of a material to transmit heat energy in a diffusive manner as given by Fourier's law

J = −kappa grad(T)

where J is the heat flux in units of energy per area per time and grad(T) is the spatial gradient of temperature. The thermal conductivity thus has units of energy per distance per time per degree K and is often approximated as an isotropic quantity, i.e. as a scalar.

The first method is to setup two thermostatted regions at opposite ends of a simulation box, or one in the middle and one at the end of a periodic box. By holding the two regions at different temperatures with a thermostatting fix, the energy added to the hot region should equal the energy subtracted from the cold region and be proportional to the heat flux moving between the regions. See the paper by Ikeshoji and Hafskjold for details of this idea. Note that thermostatting fixes such as fix nvt, fix langevin, and fix temp/rescale store the cumulative energy they add/subtract. Alternatively, the fix heat command can used in place of thermostats on each of two regions, and the resulting temperatures of the two regions monitored with the "compute temp/region" command or the temperature profile of the intermediate region monitored with the fix ave/spatial and compute ke/atom commands.

The second method is to perform a reverse non−equilibrium MD simulation using the fix thermal/conductivity command which implements the rNEMD algorithm of Muller−Plathe. Kinetic energy is swapped between atoms in two different layers of the simulation box. This induces a temperature gradient between the two layers which can be monitored with the fix ave/spatial and compute ke/atom commands. The fix tallies the cumulative energy transfer that it performs. See the fix thermal/conductivity command for details.

The third method is based on the Green−Kubo (GK) formula which relates the ensemble average of the auto−correlation of the heat flux to kappa. The heat flux can be calculated from the fluctuations of per−atom potential and kinetic energies and per−atom stress tensor in a steady−state equilibrated simulation. This is in contrast to the two preceding non−equilibrium methods, where energy flows continuously between hot and cold regions of the simulation box.

The compute heat/flux command can calculate the needed heat flux and describes how to implement the Green_Kubo formalism using additional LAMMPS commands, such as the fix ave/correlate command to calculate the needed auto−correlation. See the doc page for the compute heat/flux command for an example input script that calculates the thermal conductivity of solid Ar via the GK formalism.

6.21 Calculating viscosity

The shear viscosity eta of a fluid can be measured in at least 3 ways using various options in LAMMPS. (See this section of the manual for an analogous discussion for thermal conductivity). Eta is a measure of the propensity of a fluid to transmit momentum in a direction perpendicular to the direction of velocity or momentum flow. Alternatively it is the resistance the fluid has to being sheared. It is given by

J = −eta grad(Vstream)

where J is the momentum flux in units of momentum per area per time. and grad(Vstream) is the spatial gradient of the velocity of the fluid moving in another direction, normal to the area through which the momentum flows. Viscosity thus has units of pressure−time.

The first method is to perform a non−equlibrium MD (NEMD) simulation by shearing the simulation box via the fix deform command, and using the fix nvt/sllod command to thermostat the fluid via the SLLOD equations of motion. The velocity profile setup in the fluid by this procedure can be monitored by the fix ave/spatial command, which determines grad(Vstream) in the equation above. E.g. the derivative in the y−direction of the Vx component of fluid motion or grad(Vstream) = dVx/dy. In this case, the Pxy off−diagonal component of the pressure or stress tensor, as calculated by the compute pressure command, can also be monitored, which is the J term in the equation above. See this section of the manual for details on NEMD simulations.

The second method is to perform a reverse non−equilibrium MD simulation using the fix viscosity command which implements the rNEMD algorithm of Muller−Plathe. Momentum in one dimension is swapped between atoms in two different layers of the simulation box in a different dimension. This induces a velocity gradient which can be monitored with the fix ave/spatial command. The fix tallies the cummulative momentum transfer that it performs. See the fix viscosity command for details.

The third method is based on the Green−Kubo (GK) formula which relates the ensemble average of the auto−correlation of the stress/pressure tensor to eta. This can be done in a steady−state equilibrated simulation which is in contrast to the two preceding non−equilibrium methods, where momentum flows continuously through the simulation box.

Here is an example input script that calculates the viscosity of liquid Ar via the GK formalism:

```
# Sample LAMMPS input script for viscosity of liquid Ar
```

```
units       real
variable    T equal 86.4956
variable    V equal vol
variable    dt equal 4.0
variable    p equal 400     # correlation length
variable    s equal 5       # sample interval
variable    d equal $p*$s   # dump interval

# convert from LAMMPS real units to SI

variable    kB equal 1.3806504e-23    # [J/K/ Boltzmann
variable    atm2Pa equal 101325.0
variable    A2m equal 1.0e-10
variable    fs2s equal 1.0e-15
variable    convert equal ${atm2Pa}*${atm2Pa}*${fs2s}*${A2m}*${A2m}*${A2m}

# setup problem

dimension    3
boundary     p p p
lattice      fcc 5.376 orient x 1 0 0 orient y 0 1 0 orient z 0 0 1
region       box block 0 4 0 4 0 4
create_box   1 box
create_atoms 1 box
mass         1 39.948
pair_style   lj/cut 13.0
pair_coeff   * * 0.2381 3.405
timestep     ${dt}
thermo       $d

# equilibration and thermalization

velocity     all create $T 102486 mom yes rot yes dist gaussian
fix          NVT all nvt temp $T $T 10 drag 0.2
run          8000

# viscosity calculation, switch to NVE if desired

#unfix       NVT
#fix         NVE all nve

reset_timestep 0
variable     pxy equal pxy
variable     pxz equal pxz
variable     pyz equal pyz
fix          SS all ave/correlate $s $p $d &             v_pxy v_pxz v_pyz type auto file S0St.dat a
variable     scale equal ${convert}/(${kB}*$T)*$V*$s*${dt}
variable     v11 equal trap(f_SS[3/)*${scale}
variable     v22 equal trap(f_SS[4/)*${scale}
variable     v33 equal trap(f_SS[5/)*${scale}
thermo_style custom step temp press v_pxy v_pxz v_pyz v_v11 v_v22 v_v33
run          100000
variable     v equal (v_v11+v_v22+v_v33)/3.0
variable     ndens equal count(all)/vol
print        "average viscosity: $v [Pa.s/ @ $T K, ${ndens} /A^3"
```

(Berendsen) Berendsen, Grigera, Straatsma, J Phys Chem, 91, 6269−6271 (1987).

(Cornell) Cornell, Cieplak, Bayly, Gould, Merz, Ferguson, Spellmeyer, Fox, Caldwell, Kollman, JACS 117, 5179−5197 (1995).

(Horn) Horn, Swope, Pitera, Madura, Dick, Hura, and Head−Gordon, J Chem Phys, 120, 9665 (2004).

(Ikeshoji) Ikeshoji and Hafskjold, Molecular Physics, 81, 251−261 (1994).

(MacKerell) MacKerell, Bashford, Bellott, Dunbrack, Evanseck, Field, Fischer, Gao, Guo, Ha, et al, J Phys Chem, 102, 3586 (1998).

(Mayo) Mayo, Olfason, Goddard III, J Phys Chem, 94, 8897−8909 (1990).

(Jorgensen) Jorgensen, Chandrasekhar, Madura, Impey, Klein, J Chem Phys, 79, 926 (1983).

(Price) Price and Brooks, J Chem Phys, 121, 10096 (2004).

(Shinoda) Shinoda, Shiga, and Mikami, Phys Rev B, 69, 134103 (2004).

7. Example problems

The LAMMPS distribution includes an examples sub−directory with several sample problems. Each problem is in a sub−directory of its own. Most are 2d models so that they run quickly, requiring at most a couple of minutes to run on a desktop machine. Each problem has an input script (in.*) and produces a log file (log.*) and dump file (dump.*) when it runs. Some use a data file (data.*) of initial coordinates as additional input. A few sample log file outputs on different machines and different numbers of processors are included in the directories to compare your answers to. E.g. a log file like log.crack.foo.P means it ran on P processors of machine "foo".

The dump files produced by the example runs can be animated using the xmovie tool described in the Additional Tools section of the LAMMPS documentation. Animations of many of these examples can be viewed on the Movies section of the LAMMPS WWW Site.

These are the sample problems in the examples sub−directories:

colloid	big colloid particles in a small particle solvent, 2d system
comb	models using the COMB potential
crack	crack propagation in a 2d solid
dipole	point dipolar particles, 2d system
eim	NaCl using the EIM potential
ellipse	ellipsoidal particles in spherical solvent, 2d system
flow	Couette and Poiseuille flow in a 2d channel
friction	frictional contact of spherical asperities between 2d surfaces
indent	spherical indenter into a 2d solid
meam	MEAM test for SiC and shear (same as shear examples)
melt	rapid melt of 3d LJ system
micelle	self−assembly of small lipid−like molecules into 2d bilayers
min	energy minimization of 2d LJ melt
msst	MSST shock dynamics
neb	nudged elastic band (NEB) calculation for barrier finding
nemd	non−equilibrium MD of 2d sheared system
obstacle	flow around two voids in a 2d channel
peptide	dynamics of a small solvated peptide chain (5−mer)
peri	Peridynamic model of cylinder impacted by indenter
pour	pouring of granular particles into a 3d box, then chute flow
prd	parallel replica dynamics of a vacancy diffusion in bulk Si
reax	RDX and TATB models using the ReaxFF
rigid	rigid bodies modeled as independent or coupled

shear	sideways shear applied to 2d solid, with and without a void
srd	stochastic rotation dynamics (SRD) particles as solvent

Here is how you might run and visualize one of the sample problems:

```
cd indent
cp ../../src/lmp_linux .           # copy LAMMPS executable to this dir
lmp_linux <in.indent              # run the problem
```

Running the simulation produces the files *dump.indent* and *log.lammps*. You can visualize the dump file as follows:

```
../../tools/xmovie/xmovie −scale dump.indent
```

There is also an ELASTIC directory with an example script for computing elastic constants, using a zero temperature Si example. See the in.elastic file for more info.

There is also a USER directory which contains subdirectories of user−provided examples for user packages. See the README files in those directories for more info. See the doc/Section_start.html file for more info about user packages.

8. Performance &scalability

LAMMPS performance on several prototypical benchmarks and machines is discussed on the Benchmarks page of the LAMMPS WWW Site where CPU timings and parallel efficiencies are listed. Here, the benchmarks are described briefly and some useful rules of thumb about their performance are highlighted.

These are the 5 benchmark problems:

1. LJ = atomic fluid, Lennard−Jones potential with 2.5 sigma cutoff (55 neighbors per atom), NVE integration
2. Chain = bead−spring polymer melt of 100−mer chains, FENE bonds and LJ pairwise interactions with a 2^(1/6) sigma cutoff (5 neighbors per atom), NVE integration
3. EAM = metallic solid, Cu EAM potential with 4.95 Angstrom cutoff (45 neighbors per atom), NVE integration
4. Chute = granular chute flow, frictional history potential with 1.1 sigma cutoff (7 neighbors per atom), NVE integration
5. Rhodo = rhodopsin protein in solvated lipid bilayer, CHARMM force field with a 10 Angstrom LJ cutoff (440 neighbors per atom), particle−particle particle−mesh (PPPM) for long−range Coulombics, NPT integration

The input files for running the benchmarks are included in the LAMMPS distribution, as are sample output files. Each of the 5 problems has 32,000 atoms and runs for 100 timesteps. Each can be run as a serial benchmarks (on one processor) or in parallel. In parallel, each benchmark can be run as a fixed−size or scaled−size problem. For fixed−size benchmarking, the same 32K atom problem is run on various numbers of processors. For scaled−size benchmarking, the model size is increased with the number of processors. E.g. on 8 processors, a 256K−atom problem is run; on 1024 processors, a 32−million atom problem is run, etc.

A useful metric from the benchmarks is the CPU cost per atom per timestep. Since LAMMPS performance scales roughly linearly with problem size and timesteps, the run time of any problem using the same model (atom style, force field, cutoff, etc) can then be estimated. For example, on a 1.7 GHz Pentium desktop machine (Intel icc compiler under Red Hat Linux), the CPU run−time in seconds/atom/timestep for the 5 problems is

Problem:	LJ	Chain	EAM	Chute	Rhodopsin
CPU/atom/step:	4.55E−6	2.18E−6	9.38E−6	2.18E−6	1.11E−4
Ratio to LJ:	1.0	0.48	2.06	0.48	24.5

The ratios mean that if the atomic LJ system has a normalized cost of 1.0, the bead−spring chains and granular systems run 2x faster, while the EAM metal and solvated protein models run 2x and 25x slower respectively. The bulk of these cost differences is due to the expense of computing a particular pairwise force field for a given number of neighbors per atom.

Performance on a parallel machine can also be predicted from the one−processor timings if the parallel efficiency can be estimated. The communication bandwidth and latency of a particular parallel machine affects the efficiency. On most machines LAMMPS will give fixed−size parallel efficiencies on these benchmarks above 50% so long as the atoms/processor count is a few 100 or greater − i.e. on 64 to 128 processors. Likewise, scaled−size parallel efficiencies will typically be 80% or greater up to very large processor counts. The benchmark data on the LAMMPS WWW Site gives specific examples on some different machines, including a run of 3/4 of a billion LJ atoms on 1500 processors that ran at 85% parallel efficiency.

9. Additional tools

LAMMPS is designed to be a computational kernel for performing molecular dynamics computations. Additional pre− and post−processing steps are often necessary to setup and analyze a simulation. A few additional tools are provided with the LAMMPS distribution and are described in this section.

Our group has also written and released a separate toolkit called Pizza.py which provides tools for doing setup, analysis, plotting, and visualization for LAMMPS simulations. Pizza.py is written in Python and is available for download from the Pizza.py WWW site.

Note that many users write their own setup or analysis tools or use other existing codes and convert their output to a LAMMPS input format or vice versa. The tools listed here are included in the LAMMPS distribution as examples of auxiliary tools. Some of them are not actively supported by Sandia, as they were contributed by LAMMPS users. If you have problems using them, we can direct you to the authors.

The source code for each of these codes is in the tools sub−directory of the LAMMPS distribution. There is a Makefile (which you may need to edit for your platform) which will build several of the tools which reside in that directory. Some of them are larger packages in their own sub−directories with their own Makefiles.

- amber2lmp
- binary2txt
- ch2lmp
- chain
- createatoms
- data2xmovie
- eam database
- eam generate
- eff
- emacs
- ipp
- lmp2arc
- lmp2cfg
- lmp2vmd
- matlab
- micelle2d
- msi2lmp
- pymol_asphere
- python
- reax
- restart2data
- thermo_extract
- vim
- xmovie

amber2lmp tool

The amber2lmp sub−directory contains two Python scripts for converting files back−and−forth between the AMBER MD code and LAMMPS. See the README file in amber2lmp for more information.

These tools were written by Keir Novik while he was at Queen Mary University of London. Keir is no longer there and cannot support these tools which are out−of−date with respect to the current LAMMPS version (and maybe with respect to AMBER as well). Since we don't use these tools at Sandia, you'll need to experiment with them and make necessary modifications yourself.

---

binary2txt tool

The file binary2txt.cpp converts one or more binary LAMMPS dump file into ASCII text files. The syntax for running the tool is

```
binary2txt file1 file2 ...
```

which creates file1.txt, file2.txt, etc. This tool must be compiled on a platform that can read the binary file created by a LAMMPS run, since binary files are not compatible across all platforms.

---

ch2lmp tool

The ch2lmp sub−directory contains tools for converting files back−and−forth between the CHARMM MD code and LAMMPS.

They are intended to make it easy to use CHARMM as a builder and as a post−processor for LAMMPS. Using charmm2lammps.pl, you can convert an ensemble built in CHARMM into its LAMMPS equivalent. Using lammps2pdb.pl you can convert LAMMPS atom dumps into pdb files.

See the README file in the ch2lmp sub−directory for more information.

These tools were created by Pieter in't Veld (pjintve at sandia.gov) and Paul Crozier (pscrozi at sandia.gov) at Sandia.

---

chain tool

The file chain.f creates a LAMMPS data file containing bead−spring polymer chains and/or monomer solvent atoms. It uses a text file containing chain definition parameters as an input. The created chains and solvent atoms can strongly overlap, so LAMMPS needs to run the system initially with a "soft" pair potential to un−overlap it. The syntax for running the tool is

```
chain <def.chain > data.file
```

See the def.chain or def.chain.ab files in the tools directory for examples of definition files. This tool was used to create the system for the chain benchmark.

---

createatoms tool

The tools/createatoms directory contains a Fortran program called createAtoms.f which can generate a variety of interesting crystal structures and geometries and output the resulting list of atom coordinates in LAMMPS or other formats.

See the included Manual.pdf for details.

The tool is authored by Xiaowang Zhou (Sandia), xzhou at sandia.gov.

---

data2xmovie tool

The file data2xmovie.c converts a LAMMPS data file into a snapshot suitable for visualizing with the xmovie tool, as if it had been output with a dump command from LAMMPS itself. The syntax for running the tool is

```
data2xmovie options <infile > outfile
```

See the top of the data2xmovie.c file for a discussion of the options.

---

eam database tool

The tools/eam_database directory contains a Fortran program that will generate EAM alloy setfl potential files for any combination of 16 elements: Cu, Ag, Au, Ni, Pd, Pt, Al, Pb, Fe, Mo, Ta, W, Mg, Co, Ti, Zr. The files can then be used with the pair_style eam/alloy command.

The tool is authored by Xiaowang Zhou (Sandia), xzhou at sandia.gov, and is based on his paper:

X. W. Zhou, R. A. Johnson, and H. N. G. Wadley, Phys. Rev. B, 69, 144113 (2004).

---

eam generate tool

The tools/eam_generate directory contains several one−file C programs that convert an analytic formula into a tabulated embedded atom method (EAM) setfl potential file. The potentials they produce are in the potentials directory, and can be used with the pair_style eam/alloy command.

The source files and potentials were provided by Gerolf Ziegenhain (gerolf at ziegenhain.com).

---

eff tool

The tools/eff directory contains various scripts for generating structures and post−processing output for simulations using the electron force field (eFF).

These tools were provided by Andres Jaramillo−Botero at CalTech (ajaramil at wag.caltech.edu).

---

emacs tool

The tools/emacs directory contains a Lips add−on file for Emacs that enables a lammps−mode for editing of input scripts when using Emacs, with various highlighting options setup.

These tools were provided by Aidan Thompson at Sandia (athomps at sandia.gov).

---

ipp tool

The tools/ipp directory contains a Perl script ipp which can be used to facilitate the creation of a complicated file (say, a lammps input script or tools/createatoms input file) using a template file.

ipp was created and is maintained by Reese Jones (Sandia), rjones at sandia.gov.

See two examples in the tools/ipp directory. One of them is for the tools/createatoms tool's input file.

---

lmp2arc tool

The lmp2arc sub−directory contains a tool for converting LAMMPS output files to the format for Accelrys' Insight MD code (formerly MSI/Biosym and its Discover MD code). See the README file for more information.

This tool was written by John Carpenter (Cray), Michael Peachey (Cray), and Steve Lustig (Dupont). John is now at the Mayo Clinic (jec at mayo.edu), but still fields questions about the tool.

This tool was updated for the current LAMMPS C++ version by Jeff Greathouse at Sandia (jagreat at sandia.gov).

---

lmp2cfg tool

The lmp2cfg sub−directory contains a tool for converting LAMMPS output files into a series of *.cfg files which can be read into the AtomEye visualizer. See the README file for more information.

This tool was written by Ara Kooser at Sandia (askoose at sandia.gov).

---

lmp2vmd tool

The lmp2vmd sub−directory contains a README.txt file that describes details of scripts and plugin support within the VMD package for visualizing LAMMPS dump files.

The VMD plugins and other supporting scripts were written by Axel Kohlmeyer (akohlmey at cmm.chem.upenn.edu) at U Penn.

---

matlab tool

The matlab sub−directory contains several MATLAB scripts for post−processing LAMMPS output. The scripts include readers for log and dump files, a reader for EAM potential files, and a converter that reads LAMMPS dump files and produces CFG files that can be visualized with the AtomEye visualizer.

See the README.pdf file for more information.

These scripts were written by Arun Subramaniyan at Purdue Univ (asubrama at purdue.edu).

---

micelle2d tool

The file micelle2d.f creates a LAMMPS data file containing short lipid chains in a monomer solution. It uses a text file containing lipid definition parameters as an input. The created molecules and solvent atoms can strongly overlap, so LAMMPS needs to run the system initially with a "soft" pair potential to un−overlap it. The syntax for running the tool is

```
micelle2d <def.micelle2d > data.file
```

See the def.micelle2d file in the tools directory for an example of a definition file. This tool was used to create the system for the micelle example.

---

msi2lmp tool

The msi2lmp sub−directory contains a tool for creating LAMMPS input data files from Accelrys' Insight MD code (formerly MSI/Biosym and its Discover MD code). See the README file for more information.

This tool was written by John Carpenter (Cray), Michael Peachey (Cray), and Steve Lustig (Dupont). John is now at the Mayo Clinic (jec at mayo.edu), but still fields questions about the tool.

This tool may be out−of−date with respect to the current LAMMPS and Insight versions. Since we don't use it at Sandia, you'll need to experiment with it yourself.

---

pymol_asphere tool

The pymol_asphere sub−directory contains a tool for converting a LAMMPS dump file that contains orientation info for ellipsoidal particles into an input file for the PyMol visualization package.

Specifically, the tool triangulates the ellipsoids so they can be viewed as true ellipsoidal particles within PyMol. See the README and examples directory within pymol_asphere for more information.

This tool was written by Mike Brown at Sandia.

---

python tool

The python sub−directory contains several Python scripts that perform common LAMMPS post−processing tasks, such as:

- extract thermodynamic info from a log file as columns of numbers
- plot two columns of thermodynamic info from a log file using GnuPlot
- sort the snapshots in a dump file by atom ID
- convert multiple NEB dump files into one dump file for viz
- convert dump files into XYZ, CFG, or PDB format for viz by other packages

These are simple scripts built on Pizza.py modules. See the README for more info on Pizza.py and how to use these scripts.

---

reax tool

The reax sub−directory contains stand−alond codes that can post−process the output of the fix reax/bonds command from a LAMMPS simulation using ReaxFF. See the README.txt file for more info.

These tools were written by Aidan Thompson at Sandia.

---

restart2data tool

The file restart2data.cpp converts a binary LAMMPS restart file into an ASCII data file. The syntax for running the tool is

```
restart2data restart−file data−file (input−file)
```

Input−file is optional and if specified will contain LAMMPS input commands for the masses and force field parameters, instead of putting those in the data−file. Only a few force field styles currently support this option.

This tool must be compiled on a platform that can read the binary file created by a LAMMPS run, since binary files are not compatible across all platforms.

Note that a text data file has less precision than a binary restart file. Hence, continuing a run from a converted data file will typically not conform as closely to a previous run as will restarting from a binary restart file.

If a "%" appears in the specified restart−file, the tool expects a set of multiple files to exist. See the restart and write_restart commands for info on how such sets of files are written by LAMMPS, and how the files are named.

---

thermo_extract tool

The thermo_extract tool reads one of more LAMMPS log files and extracts a thermodynamic value (e.g. Temp, Press). It spits out the time,value as 2 columns of numbers so the tool can be used as a quick way to plot some quantity of interest. See the header of the thermo_extract.c file for the syntax of how to run it and other details.

This tool was written by Vikas Varshney at Wright Patterson AFB (vikas.varshney at gmail.com).

---

vim tool

The files in the tools/vim directory are add−ons to the VIM editor that allow easier editing of LAMMPS input scripts. See the README.txt file for details.

These files were provided by Gerolf Ziegenhain (gerolf at ziegenhain.com)

---

xmovie tool

The xmovie tool is an X−based visualization package that can read LAMMPS dump files and animate them. It is in its own sub−directory with the tools directory. You may need to modify its Makefile so that it can find the appropriate X libraries to link against.

The syntax for running xmovie is

```
xmovie options dump.file1 dump.file2 ...
```

If you just type "xmovie" you will see a list of options. Note that by default, LAMMPS dump files are in scaled coordinates, so you typically need to use the −scale option with xmovie. When xmovie runs it opens a visualization window and a control window. The control options are straightforward to use.

Xmovie was mostly written by Mike Uttormark (U Wisconsin) while he spent a summer at Sandia. It displays 2d projections of a 3d domain. While simple in design, it is an amazingly fast program that can render large numbers of atoms very quickly. It's a useful tool for debugging LAMMPS input and output and making sure your simulation is doing what you think it should. The animations on the Examples page of the LAMMPS WWW site were created with xmovie.

I've lost contact with Mike, so I hope he's comfortable with us distributing his great tool!

10. Modifying &extending LAMMPS

LAMMPS is designed in a modular fashion so as to be easy to modify and extend with new functionality. In fact, about 75% of its source code is files added in this fashion.

In this section, changes and additions users can make are listed along with minimal instructions. If you add a new feature to LAMMPS and think it will be of interest to general users, we encourage you to submit it to the developers for inclusion in the released version of LAMMPS. Information about how to do this is provided below.

The best way to add a new feature is to find a similar feature in LAMMPS and look at the corresponding source and header files to figure out what it does. You will need some knowledge of C++ to be able to understand the hi−level structure of LAMMPS and its class organization, but functions (class methods) that do actual computations are written in vanilla C−style code and operate on simple C−style data structures (vectors and arrays).

Most of the new features described in this section require you to write a new C++ derived class (except for exceptions described below, where you can make small edits to existing files). Creating a new class requires 2 files, a source code file (*.cpp) and a header file (*.h). The derived class must provide certain methods to work as a new option. Depending on how different your new feature is compared to existing features, you can either derive from the base class itself, or from a derived class that already exists. Enabling LAMMPS to invoke the new class is as simple as putting the two source files in the src dir and re−building LAMMPS.

The advantage of C++ and its object−orientation is that all the code and variables needed to define the new feature are in the 2 files you write, and thus shouldn't make the rest of LAMMPS more complex or cause side−effect bugs.

Here is a concrete example. Suppose you write 2 files pair_foo.cpp and pair_foo.h that define a new class PairFoo that computes pairwise potentials described in the classic 1997 paper by Foo, et al. If you wish to invoke those potentials in a LAMMPS input script with a command like

```
pair_style foo 0.1 3.5
```

then your pair_foo.h file should be structured as follows:

```
#ifdef PAIR_CLASS
PairStyle(foo,PairFoo)
#else
...
(class definition for PairFoo)
...
#endif
```

where "foo" is the style keyword in the pair_style command, and PairFoo is the class name defined in your pair_foo.cpp and pair_foo.h files.

When you re−build LAMMPS, your new pairwise potential becomes part of the executable and can be invoked with a pair_style command like the example above. Arguments like 0.1 and 3.5 can be defined and processed by your new class.

As illustrated by this pairwise example, many kinds of options are referred to in the LAMMPS documentation as the "style" of a particular command.

The instructions below give the header file for the base class that these styles are derived from. Public variables in that file are ones used and set by the derived classes which are also used by the base class. Sometimes they are also used by the rest of LAMMPS. Virtual functions in the base class header file which are set = 0 are ones you must define in your new derived class to give it the functionality LAMMPS expects. Virtual functions that are not set to 0 are functions you can optionally define.

Additionally, new output options can be added directly to the thermo.cpp, dump_custom.cpp, and variable.cpp files as explained below.

Here are additional guidelines for modifying LAMMPS and adding new functionality:

- Think about whether what you want to do would be better as a pre− or post−processing step. Many computations are more easily and more quickly done that way.
- Don't do anything within the timestepping of a run that isn't parallel. E.g. don't accumulate a bunch of data on a single processor and analyze it. You run the risk of seriously degrading the parallel efficiency.
- If your new feature reads arguments or writes output, make sure you follow the unit conventions discussed by the units command.
- If you add something you think is truly useful and doesn't impact LAMMPS performance when it isn't used, send an email to the developers. We might be interested in adding it to the LAMMPS distribution. See further details on this at the bottom of this page.

Here are the subsequent topics discussed below, most of which are new features that can be added in the manner just described:

10.1 Atom styles
10.2 Bond, angle, dihedral, improper potentials
10.3 Compute styles
10.4 Dump styles
10.5 Dump custom output options
10.6 Fix styles which include integrators, temperature and pressure control, force constraints, boundary conditions, diagnostic output, etc
10.7 Input script commands
10.8 Kspace computations
10.9 Minimization styles
10.10 Pairwise potentials
10.11 Region styles
10.12 Thermodynamic output options
10.13 Variable options
10.14 Submitting new features for inclusion in LAMMPS

10.1 Atom styles

Classes that define an atom style are derived from the Atom class. The atom style determines what quantities are associated with an atom. A new atom style can be created if one of the existing atom styles does not define all the arrays you need to store and communicate with atoms.

Atom_vec_atomic.cpp is a simple example of an atom style.

Here is a brief description of methods you define in your new derived class. See atom.h for details.

grow	re−allocate atom arrays to longer lengths
copy	copy info for one atom to another atom's array locations
pack_comm	store an atom's info in a buffer communicated every timestep
pack_comm_vel	add velocity info to buffer
pack_comm_one	store extra info unique to this atom style
unpack_comm	retrieve an atom's info from the buffer
unpack_comm_vel	also retrieve velocity info
unpack_comm_one	retreive extra info unique to this atom style
pack_reverse	store an atom's info in a buffer communicating partial forces
pack_reverse_one	store extra info unique to this atom style
unpack_reverse	retrieve an atom's info from the buffer
unpack_reverse_one	retreive extra info unique to this atom style
pack_border	store an atom's info in a buffer communicated on neighbor re−builds
pack_border_vel	add velocity info to buffer
pack_border_one	store extra info unique to this atom style
unpack_border	retrieve an atom's info from the buffer
unpack_border_vel	also retrieve velocity info
unpack_border_one	retreive extra info unique to this atom style
pack_exchange	store all an atom's info to migrate to another processor
unpack_exchange	retrieve an atom's info from the buffer
size_restart	number of restart quantities associated with proc's atoms
pack_restart	pack atom quantities into a buffer
unpack_restart	unpack atom quantities from a buffer
create_atom	create an individual atom of this style
data_atom	parse an atom line from the data file
memory_usage	tally memory allocated by atom arrays

The constructor of the derived class sets values for several variables that you must set when defining a new atom style, which are documented in atom_vec.h. New atom arrays are defined in atom.cpp. Search for the word "customize" and you will find locations you will need to modify.

10.2 Bond, angle, dihedral, improper potentials

Classes that compute molecular interactions are derived from the Bond, Angle, Dihedral, and Improper classes. New styles can be created to add new potentials to LAMMPS.

Bond_harmonic.cpp is the simplest example of a bond style. Ditto for the harmonic forms of the angle, dihedral, and improper style commands.

Here is a brief description of methods you define in your new derived bond class. See bond.h, angle.h, dihedral.h, and improper.h for details.

compute	compute the molecular interactions
coeff	

	set coefficients for one bond type
equilibrium_distance	length of bond, used by SHAKE
write &read_restart	writes/reads coeffs to restart files
single	force and energy of a single bond

10.3 Compute styles

Classes that compute scalar and vector quantities like temperature and the pressure tensor, as well as classes that compute per−atom quantities like kinetic energy and the centro−symmetry parameter are derived from the Compute class. New styles can be created to add new calculations to LAMMPS.

Compute_temp.cpp is a simple example of computing a scalar temperature. Compute_ke_atom.cpp is a simple example of computing per−atom kinetic energy.

Here is a brief description of methods you define in your new derived class. See compute.h for details.

compute_scalar	compute a scalar quantity
compute_vector	compute a vector of quantities
compute_peratom	compute one or more quantities per atom
pack_comm	pack a buffer with items to communicate
unpack_comm	unpack the buffer
pack_reverse	pack a buffer with items to reverse communicate
unpack_reverse	unpack the buffer
memory_usage	tally memory usage

10.4 Dump styles

10.5 Dump custom output options

Classes that dump per−atom info to files are derived from the Dump class. To dump new quantities or in a new format, a new derived dump class can be added, but it is typically simpler to modify the DumpCustom class contained in the dump_custom.cpp file.

Dump_atom.cpp is a simple example of a derived dump class.

Here is a brief description of methods you define in your new derived class. See dump.h for details.

write_header	write the header section of a snapshot of atoms
count	count the number of lines a processor will output
pack	pack a proc's output data into a buffer
write_data	write a proc's data to a file

See the dump command and its *custom* style for a list of keywords for atom information that can already be dumped by DumpCustom. It includes options to dump per−atom info from Compute classes, so adding a new derived Compute class is one way to calculate new quantities to dump.

Alternatively, you can add new keywords to the dump custom command. Search for the word "customize" in dump_custom.cpp to see the half−dozen or so locations where code will need to be added.

10.6 Fix styles

In LAMMPS, a "fix" is any operation that is computed during timestepping that alters some property of the system. Essentially everything that happens during a simulation besides force computation, neighbor list construction, and output, is a "fix". This includes time integration (update of coordinates and velocities), force constraints or boundary conditions (SHAKE or walls), and diagnostics (compute a diffusion coefficient). New styles can be created to add new options to LAMMPS.

Fix_setforce.cpp is a simple example of setting forces on atoms to prescribed values. There are dozens of fix options already in LAMMPS; choose one as a template that is similar to what you want to implement.

Here is a brief description of methods you can define in your new derived class. See fix.h for details.

setmask	determines when the fix is called during the timestep
init	initialization before a run
setup	called immediately before the 1st timestep
initial_integrate	called at very beginning of each timestep
pre_exchange	called before atom exchange on re−neighboring steps
pre_neighbor	called before neighbor list build
post_force	called after pair &molecular forces are computed
final_integrate	called at end of each timestep
end_of_step	called at very end of timestep
write_restart	dumps fix info to restart file
restart	uses info from restart file to re−initialize the fix
grow_arrays	allocate memory for atom−based arrays used by fix
copy_arrays	copy atom info when an atom migrates to a new processor
memory_usage	report memory used by fix
pack_exchange	store atom's data in a buffer
unpack_exchange	retrieve atom's data from a buffer
pack_restart	store atom's data for writing to restart file
unpack_restart	retrieve atom's data from a restart file buffer
size_restart	size of atom's data
maxsize_restart	max size of atom's data
initial_integrate_respa	same as initial_integrate, but for rRESPA
post_force_respa	same as post_force, but for rRESPA
final_integrate_respa	same as final_integrate, but for rRESPA
pack_comm	pack a buffer to communicate a per−atom quantity
unpack_comm	unpack a buffer to communicate a per−atom quantity

pack_reverse_comm	pack a buffer to reverse communicate a per−atom quantity
unpack_reverse_comm	unpack a buffer to reverse communicate a per−atom quantity
thermo	compute quantities for thermodynamic output

Typically, only a small fraction of these methods are defined for a particular fix. Setmask is mandatory, as it determines when the fix will be invoked during the timestep. Fixes that perform time integration (*nve*, *nvt*, *npt*) implement initial_integrate() and final_integrate() to perform velocity Verlet updates. Fixes that constrain forces implement post_force().

Fixes that perform diagnostics typically implement end_of_step(). For an end_of_step fix, one of your fix arguments must be the variable "nevery" which is used to determine when to call the fix and you must set this variable in the constructor of your fix. By convention, this is the first argument the fix defines (after the ID, group−ID, style).

If the fix needs to store information for each atom that persists from timestep to timestep, it can manage that memory and migrate the info with the atoms as they move from processors to processor by implementing the grow_arrays, copy_arrays, pack_exchange, and unpack_exchange methods. Similarly, the pack_restart and unpack_restart methods can be implemented to store information about the fix in restart files. If you wish an integrator or force constraint fix to work with rRESPA (see the run_style command), the initial_integrate, post_force_integrate, and final_integrate_respa methods can be implemented. The thermo method enables a fix to contribute values to thermodynamic output, as printed quantities and/or to be summed to the potential energy of the system.

10.7 Input script commands

New commands can be added to LAMMPS input scripts by adding new classes that have a "command" method. For example, the create_atoms, read_data, velocity, and run commands are all implemented in this fashion. When such a command is encountered in the LAMMPS input script, LAMMPS simply creates a class with the corresponding name, invokes the "command" method of the class, and passes it the arguments from the input script. The command method can perform whatever operations it wishes on LAMMPS data structures.

The single method your new class must define is as follows:

command	operations performed by the new command

Of course, the new class can define other methods and variables as needed.

10.8 Kspace computations

Classes that compute long−range Coulombic interactions via K−space representations (Ewald, PPPM) are derived from the KSpace class. New styles can be created to add new K−space options to LAMMPS.

Ewald.cpp is an example of computing K−space interactions.

Here is a brief description of methods you define in your new derived class. See kspace.h for details.

init	initialize the calculation before a run
setup	computation before the 1st timestep of a run

compute	every−timestep computation
memory_usage	tally of memory usage

10.9 Minimization styles

Classes that perform energy minimization derived from the Min class. New styles can be created to add new minimization algorithms to LAMMPS.

Min_cg.cpp is an example of conjugate gradient minimization.

Here is a brief description of methods you define in your new derived class. See min.h for details.

init	initialize the minimization before a run
run	perform the minimization
memory_usage	tally of memory usage

10.10 Pairwise potentials

Classes that compute pairwise interactions are derived from the Pair class. In LAMMPS, pairwise calculation include manybody potentials such as EAM or Tersoff where particles interact without a static bond topology. New styles can be created to add new pair potentials to LAMMPS.

Pair_lj_cut.cpp is a simple example of a Pair class, though it includes some optional methods to enable its use with rRESPA.

Here is a brief description of the class methods in pair.h:

compute	workhorse routine that computes pairwise interactions
settings	reads the input script line with arguments you define
coeff	set coefficients for one i,j type pair
init_one	perform initialization for one i,j type pair
init_style	initialization specific to this pair style
write &read_restart	write/read i,j pair coeffs to restart files
write &read_restart_settings	write/read global settings to restart files
single	force and energy of a single pairwise interaction between 2 atoms
compute_inner/middle/outer	versions of compute used by rRESPA

The inner/middle/outer routines are optional.

10.11 Region styles

Classes that define geometric regions are derived from the Region class. Regions are used elsewhere in LAMMPS to group atoms, delete atoms to create a void, insert atoms in a specified region, etc. New styles can be created to add new region shapes to LAMMPS.

Region_sphere.cpp is an example of a spherical region.

Here is a brief description of methods you define in your new derived class. See region.h for details.

match	determine whether a point is in the region

10.12 Thermodynamic output options

There is one class that computes and prints thermodynamic information to the screen and log file; see the file thermo.cpp.

There are two styles defined in thermo.cpp: "one" and "multi". There is also a flexible "custom" style which allows the user to explicitly list keywords for quantities to print when thermodynamic info is output. See the thermo_style command for a list of defined quantities.

The thermo styles (one, multi, etc) are simply lists of keywords. Adding a new style thus only requires defining a new list of keywords. Search for the word "customize" with references to "thermo style" in thermo.cpp to see the two locations where code will need to be added.

New keywords can also be added to thermo.cpp to compute new quantities for output. Search for the word "customize" with references to "keyword" in thermo.cpp to see the several locations where code will need to be added.

Note that the thermo_style custom command already allows for thermo output of quantities calculated by fixes, computes, and variables. Thus, it may be simpler to compute what you wish via one of those constructs, than by adding a new keyword to the thermo command.

10.13 Variable options

There is one class that computes and stores variable information in LAMMPS; see the file variable.cpp. The value associated with a variable can be periodically printed to the screen via the print, fix print, or thermo_style custom commands. Variables of style "equal" can compute complex equations that involve the following types of arguments:

thermo keywords = ke, vol, atoms, ... other variables = v_a, v_myvar, ... math functions = div(x,y), mult(x,y), add(x,y), ... group functions = mass(group), xcm(group,x), ... atom values = x**123**, y**3**, vx**34**, ... compute values = c_mytemp**0**, c_thermo_press**3**, ...

Adding keywords for the thermo_style custom command (which can then be accessed by variables) was discussed here on this page.

Adding a new math function of one or two arguments can be done by editing one section of the Variable::evaulate() method. Search for the word "customize" to find the appropriate location.

Adding a new group function can be done by editing one section of the Variable::evaulate() method. Search for the word "customize" to find the appropriate location. You may need to add a new method to the Group class as well (see the group.cpp file).

Accessing a new atom−based vector can be done by editing one section of the Variable::evaulate() method. Search for the word "customize" to find the appropriate location.

Adding new compute styles (whose calculated values can then be accessed by variables) was discussed here on this page.

10.14 Submitting new features for inclusion in LAMMPS

We encourage users to submit new features that they add to LAMMPS to the developers, especially if you think the features will be of interest to other users. If they are broadly useful we may add them as core files to LAMMPS or as part of a standard package. Else we will add them as a user−contributed package or file. Examples of user packages are in src sub−directories that start with USER. The USER−MISC package is simply a collection of (mostly) unrelated single files, which is the simplest way to have your contribution quickly added to the LAMMPS distribution. You can see a list of the both standard and user packages by typing "make package" in the LAMMPS src directory.

With user packages and files, all we are really providing (aside from the fame and fortune that accompanies having your name in the source code and on the Authors page of the LAMMPS WWW site), is a means for you to distribute your work to the LAMMPS user community and a mechanism for others to easily try out your new feature. This may help you find bugs or make contact with new collaborators. Note that you're also implicitly agreeing to support your code which means answer questions, fix bugs, and maintain it if LAMMPS changes.

The previous sections of this doc page describe how to add new features of various kinds to LAMMPS. Packages are simply collections of one or more new class files which are invoked as a new "style" within a LAMMPS input script. If designed correctly, these additions do not require changes to the main core of LAMMPS; they are simply add−on files. If you think your new feature requires non−trivial changes in core LAMMPS files, you'll need to communicate with the developers, since we may or may not want to make those changes. An example of a trivial change is making a parent−class method "virtual" when you derive a new child class from it.

Here is what you need to do to submit a user package or single file for our consideration. Following these steps will save time for both you and us. See existing package files for examples.

- All source files you provide must compile with the most current version of LAMMPS.
- If your contribution is a single file (actually a *.cpp and *.h file) it can most rapidly be added to the USER−MISC directory. Send us the one−line entry to add to the USER−MISC/README file in that dir, along with the 2 source files. You can do this multiple times if you wish to contribute several individual features.
- If your contribution is several related featues, it is probably best to make it a user package directory with a name like USER−FOO. In addition to your new files, the directory should contain a README, and Install.csh file. The README text file should contain your name and contact information and a brief description of what your new package does. The Install.csh file enables LAMMPS to include and exclude your package. See other README and Install.sh files in other USER directories as examples. Send us a tarball of this USER−FOO directory.
- Your new source files need to have the LAMMPS copyright, GPL notice, and your name at the top, like other LAMMPS source files. They need to create a class that is inside the LAMMPS namespace. Other than that, your files can do whatever is necessary to implement the new features. They don't have to be written in the same stylistic format and syntax as other LAMMPS files, though that would be nice.
- Finally, you must also send a documentation file for each new command or style you are adding to LAMMPS. This will be one file for a single−file feature. For a package, it might be several files. These are simple text files which we will convert to HTML. They must be in the same format as other *.txt files in the lammps/doc directory for similar commands and styles. The "Restrictions" section of the doc page should indicate that your command is only available if LAMMPS is built with the appropriate USER−MISC or USER−FOO package. See other user package doc files for an example of how to do this.

The txt2html tool we use to do the conversion can be downloaded from this site, so you can perform the HTML conversion yourself to proofread your doc page.

Note that the more clear and self−explanatory you make your doc and README files, the more likely it is that users will try out your new feature.

(Foo) Foo, Morefoo, and Maxfoo, J of Classic Potentials, 75, 345 (1997).

11. Python interface to LAMMPS

The LAMMPS distribution includes some Python code in its python directory which wraps the library interface to LAMMPS. This makes it is possible to run LAMMPS, invoke LAMMPS commands or give it an input script, extract LAMMPS results, an modify internal LAMMPS variables, either from a Python script or interactively from a Python prompt.

Python is a powerful scripting and programming language which can be used to wrap software like LAMMPS and other packages. It can be used to glue multiple pieces of software together, e.g. to run a coupled or multiscale model. See this section of the manual and the couple directory of the distribution for more ideas about coupling LAMMPS to other codes. See this section about how to build LAMMPS as a library, and this section for a description of the library interface provided in src/library.cpp and src/library.h and how to extend it for your needs. As described below, that interface is what is exposed to Python. It is designed to be easy to add functions to. This has the effect of extending the Python inteface as well. See details below.

By using the Python interface LAMMPS can also be coupled with a GUI or visualization tools that display graphs or animations in real time as LAMMPS runs. Examples of such scripts are inlcluded in the python directory.

Two advantages of using Python are how concise the language is and that it can be run interactively, enabling rapid development and debugging of programs. If you use it to mostly invoke costly operations within LAMMPS, such as running a simulation for a reasonable number of timesteps, then the overhead cost of invoking LAMMPS thru Python will be negligible.

Before using LAMMPS from a Python script, the Python on your machine must be "extended" to include an interface to the LAMMPS library. If your Python script will invoke MPI operations, you will also need to extend your Python with an interface to MPI itself.

Thus you should first decide how you intend to use LAMMPS from Python. There are 3 options:

(1) Use LAMMPS on a single processor running Python.

(2) Use LAMMPS in parallel, where each processor runs Python, but your Python program does not use MPI.

(3) Use LAMMPS in parallel, where each processor runs Python, and your Python script also makes MPI calls through a Python/MPI interface.

Note that for (2) and (3) you will not be able to use Python interactively by typing commands and getting a response. This is because you will have multiple instances of Python running (e.g. on a parallel machine) and they cannot all read what you type.

Working in mode (1) does not require your machine to have MPI installed. You should extend your Python with a serial version of LAMMPS and the dummy MPI library provided with LAMMPS. See instructions below on how to do this.

Working in mode (2) requires your machine to have an MPI library installed, but your Python does not need to be extended with MPI itself. The MPI library must be a shared library (e.g. a *.so file on Linux) which is not typically created when MPI is built/installed. See instruction below on how to do this. You should extend your Python with the a parallel versionn of LAMMPS which will use the shared MPI system library. See instructions below on how to do this.

Working in mode (3) requires your machine to have MPI installed (as a shared library as in (2)). You must also extend your Python with a parallel version of LAMMPS (same as in (2)) and with MPI itself, via one of several available Python/MPI packages. See instructions below on how to do the latter task.

Several of the following sub−sections cover the rest of the Python setup discussion. The next to last sub−section describes the Python syntax used to invoke LAMMPS. The last sub−section describes example Python scripts included in the python directory.

- 11.1 Extending Python with a serial version of LAMMPS
- 11.2 Creating a shared MPI library
- 11.3 Extending Python with a parallel version of LAMMPS
- 11.4 Extending Python with MPI
- 11.5 Testing the Python−LAMMPS interface
- 11.6 Using LAMMPS from Python
- 11.7 Example Python scripts that use LAMMPS

Before proceeding, there are 2 items to note.

(1) The provided Python wrapper for LAMMPS uses the amazing and magical (to me) "ctypes" package in Python, which auto−generates the interface code needed between Python and a set of C interface routines for a library. Ctypes is part of standard Python for versions 2.5 and later. You can check which version of Python you have installed, by simply typing "python" at a shell prompt.

(2) Any library wrapped by Python, including LAMMPS, must be built as a shared library (e.g. a *.so file on Linux and not a *.a file). The python/setup_serial.py and setup.py scripts do this build for LAMMPS itself (described below). But if you have LAMMPS configured to use additional packages that have their own libraries, then those libraries must also be shared libraries. E.g. MPI, FFTW, or any of the libraries in lammps/lib. When you build LAMMPS as a stand−alone code, you are not building shared versions of these libraries.

The discussion below describes how to create a shared MPI library. I suggest you start by configuing LAMMPS without packages installed that require any libraries besides MPI. See this section of the manual for a discussion of LAMMPS pacakges. E.g. do not use the KSPACE, GPU, MEAM, POEMS, or REAX packages.

If you are successfully follow the steps belwo to build the Python wrappers and use this version of LAMMPS through Python, you can then take the next step of adding LAMMPS packages that use additional libraries. This will require you to build a shared library for that package's library, similar to what is described below for MPI. It will also require you to edit the python/setup_serial.py or setup.py scripts to enable Python to access those libraries when it builds the LAMMPS wrapper.

11.1 Extending Python with a serial version of LAMMPS

From the python directory in the LAMMPS distribution, type

```
python setup_serial.py build
```

and then one of these commands:

```
sudo python setup_serial.py install
python setup_serial.py install −−home=~/foo
```

The "build" command should compile all the needed LAMMPS files, including its dummy MPI library. The first "install" command will put the needed files in your Python's site−packages sub−directory, so that Python can load them. For example, if you installed Python yourself on a Linux machine, it would typically be somewhere like /usr/local/lib/python2.5/site−packages. Installing Python packages this way often requires you to be able to write to the Python directories, which may require root priveleges, hence the "sudo" prefix. If this is not the case, you can drop the "sudo".

Alternatively, you can install the LAMMPS files (or any other Python packages) in your own user space. The second "install" command does this, where you should replace "foo" with your directory of choice.

If these commands are successful, a *lammps.py* and *_lammps_serial.so* file will be put in the appropriate directory.

11.2 Creating a shared MPI library

A shared library is one that is dynamically loadable, which is what Python requires. On Linux this is a library file that ends in ".so", not ".a". Such a shared library is normally not built if you installed MPI yourself, but it is easy to do. Here is how to do it for MPICH, a popular open−source version of MPI, distributed by Argonne National Labs. From within the mpich directory, type

```
./configure −−enable−sharedlib=gcc
make
make install
```

You may need to use "sudo make install" in place of the last line. The end result should be the file libmpich.so in /usr/local/lib.

IMPORTANT NOTE: If the file libmpich.a already exists in your installation directory (e.g. /usr/local/lib), you will now have both a static and shared MPI library. This will be fine for running LAMMPS from Python since it only uses the shared library. But if you now try to build LAMMPS by itself as a stand−alone program (cd lammps/src; make foo) or build other codes that expect to link against libmpich.a, then those builds will typically fail if the linker uses libmpich.so instead. This means you will need to remove the file /usr/local/lib/libmich.so before building LAMMPS again as a stand−alone code.

11.3 Extending Python with a parallel version of LAMMPS

From the python directory, type

```
python setup.py build
```

and then one of these commands:

```
sudo python setup.py install
python setup.py install −−home=~/foo
```

The "build" command should compile all the needed LAMMPS C++ files, which will require MPI to be installed on your system. This means it must find both the header file mpi.h and a shared library file, e.g. libmpich.so if the MPICH version of MPI is installed. See the preceding section for how to create a shared library version of MPI if it does not exist. You may need to adjust the "include_dirs" and "library_dirs" and "libraries" fields in python/setup.py to insure the Python build finds all the files it needs.

The first "install" command will put the needed files in your Python's site−packages sub−directory, so that Python can load them. For example, if you installed Python yourself on a Linux machine, it would typically be somewhere like /usr/local/lib/python2.5/site−packages. Installing Python packages this way often requires you to be able to write to the Python directories, which may require root priveleges, hence the "sudo" prefix. If this is not the case, you can drop the "sudo".

Alternatively, you can install the LAMMPS files (or any other Python packages) in your own user space. The second "install" command does this, where you should replace "foo" with your directory of choice.

If these commands are successful, a *lammps.py* and *_lammps.so* file will be put in the appropriate directory.

11.4 Extending Python with MPI

There are several Python packages available that purport to wrap MPI as a library and allow MPI functions to be called from Python.

These include

- pyMPI
- maroonmpi
- mpi4py
- myMPI
- Pypar

All of these except pyMPI work by wrapping the MPI library (which must be available on your system as a shared library, as discussed above), and exposing (some portion of) its interface to your Python script. This means they cannot be used interactively in parallel, since they do not address the issue of interactive input to multiple instances of Python running on different processors. The one exception is pyMPI, which alters the Python interpreter to address this issue, and (I believe) creates a new alternate executable (in place of python itself) as a result.

In principle any of these Python/MPI packages should work to invoke both calls to LAMMPS and MPI itself from a Python script running in parallel. However, when I downloaded and looked at a few of them, their docuemtation was incomplete and I had trouble with their installation. It's not clear if some of the packages are still being actively developed and supported.

The one I recommend, since I have successfully used it with LAMMPS, is Pypar. Pypar requires the ubiquitous Numpy package be installed in your Python. After launching python, type

```
>>> import numpy
```

to see if it is installed. If not, here is how to install it (version 1.3.0b1 as of April 2009). Unpack the numpy tarball and from its top−level directory, type

```
python setup.py build
sudo python setup.py install
```

The "sudo" is only needed if required to copy Numpy files into your Python distribution's site−packages directory.

To install Pypar (version pypar−2.1.0_66 as of April 2009), unpack it and from its "source" directory, type

```
python setup.py build
```

```
sudo python setup.py install
```

Again, the "sudo" is only needed if required to copy PyPar files into your Python distribution's site−packages directory.

If you have successully installed Pypar, you should be able to run python serially and type

```
>>> import pypar
```

without error. You should also be able to run python in parallel on a simple test script

```
% mpirun −np 4 python test.script
```

where test.script contains the lines

```
import pypar
print "Proc %d out of %d procs" % (pypar.rank(),pypar.size())
```

and see one line of output for each processor you ran on.

11.5 Testing the Python−LAMMPS interface

Before using LAMMPS in a Python program, one more step is needed. The interface to LAMMPS is via the Python ctypes package, which loads the shared LAMMPS library via a CDLL() call, which in turn is a wrapper on the C−library dlopen(). This command is different than a normal Python "import" and needs to be able to find the LAMMPS shared library, which is either in the Python site−packages directory or in a local directory you specified in the "python setup.py install" command, as described above.

The simplest way to do this is add a line like this to your .cshrc or other shell start−up file.

```
setenv LD_LIBRARY_PATH
${LD_LIBRARY_PATH}:/usr/local/lib/python2.5/site−packages
```

and then execute the shell file to insure the path has been updated. This will extend the path that dlopen() uses to look for shared libraries.

To test if the serial LAMMPS library has been successfully installed (mode 1 above), launch Python and type

```
>>> from lammps import lammps
>>> lmp = lammps()
```

If you get no errors, you're ready to use serial LAMMPS from Python.

If you built LAMMPS for parallel use (mode 2 or 3 above), launch Python in parallel:

```
% mpirun −np 4 python test.script
```

where test.script contains the lines

```
import pypar
from lammps import lammps
lmp = lammps()
print "Proc %d out of %d procs has" % (pypar.rank(),pypar.size()), lmp
pypar.finalize()
```

Again, if you get no errors, you're good to go.

Note that if you left out the "import pypar" line from this script, you would instantiate and run LAMMPS independently on each of the P processors specified in the mpirun command. You can test if Pypar is enabling true parallel Python and LAMMPS by adding a line to the above sequence of commands like lmp.file("in.lj") to run an input script and see if the LAMMPS run says it ran on P processors or if you get output from P duplicated 1−processor runs written to the screen. In the latter case, Pypar is not working correctly.

Note that this line:

```
from lammps import lammps
```

will import either the serial or parallel version of the LAMMPS library, as wrapped by lammps.py. But if you installed both via setup_serial.py and setup.py, it will always import the parallel version, since it attempts that first.

Note that if your Python script imports the Pypar package (as above), so that it can use MPI calls directly, then Pypar initializes MPI for you. Thus the last line of your Python script should be pypar.finalize(), to insure MPI is shut down correctly.

Also note that a Python script can be invoked in one of several ways:

% python foo.script % python −i foo.script % foo.script

The last command requires that the first line of the script be something like this:

#!/usr/local/bin/python #!/usr/local/bin/python −i

where the path points to where you have Python installed, and that you have made the script file executable:

% chmod +x foo.script

Without the "−i" flag, Python will exit when the script finishes. With the "−i" flag, you will be left in the Python interpreter when the script finishes, so you can type subsequent commands. As mentioned above, you can only run Python interactively when running Python on a single processor, not in parallel.

11.6 Using LAMMPS from Python

The Python interface to LAMMPS consists of a Python "lammps" module, the source code for which is in python/lammps.py, which creates a "lammps" object, with a set of methods that can be invoked on that object. The sample Python code below assumes you have first imported the "lammps" module in your Python script and its settings as follows:

```
from lammps import lammps
from lammps import LMPINT as INT
from lammps import LMPDOUBLE as DOUBLE
from lammps import LMPIPTR as IPTR
from lammps import LMPDPTR as DPTR
from lammps import LMPDPTRPTR as DPTRPTR
```

These are the methods defined by the lammps module. If you look at the file src/library.cpp you will see that they

correspond one−to−one with calls you can make to the LAMMPS library from a C++ or C or Fortran program.

```
lmp = lammps()           # create a LAMMPS object
lmp = lammps(list)       # ditto, with command−line args, list = ["−echo","screen"]

lmp.close()              # destroy a LAMMPS object

lmp.file(file)           # run an entire input script, file = "in.lj"
lmp.command(cmd)         # invoke a single LAMMPS command, cmd = "run 100"

xlo = lmp.extract_global(name,type)  # extract a global quantity
                                     # name = "boxxlo", "nlocal", etc
                                     # type = INT or DOUBLE

coords = lmp.extract_atom(name,type)      # extract a per−atom quantity
                                          # name = "x", "type", etc
                                          # type = IPTR or DPTR or DPTRPTR

eng = lmp.extract_compute(id,style,type)  # extract value(s) from a compute
v3 = lmp.extract_fix(id,style,type,i,j)   # extract value(s) from a fix
                                          # id = ID of compute or fix
                                          # style = 0 = global data
                                          #         1 = per−atom data
                                          #         2 = local data
                                          # type = 0 = scalar
                                          #        1 = vector
                                          #        2 = array
                                          # i,j = indices of value in global vector or array

var = lmp.extract_variable(name,group,flag)  # extract value(s) from a variable
                                             # name = name of variable
                                             # group = group ID (ignored for equal−style variables)
                                             # flag = 0 = equal−style variable
                                             #        1 = atom−style variable

natoms = lmp.get_natoms()                 # total # of atoms as int
x = lmp.get_coords()                      # return coords of all atoms in x
lmp.put_coords(x)                         # set all atom coords via x
```

The creation of a LAMMPS object does not take an MPI communicator as an argument. There should be a way to do this, so that the LAMMPS instance runs on a subset of processors, if desired, but I don't yet know how from Pypar. So for now, it runs on MPI_COMM_WORLD, which is all the processors.

The file() and command() methods allow an input script or single commands to be invoked.

The extract_global(), extract_atom(), extract_compute(), extract_fix(), and extract_variable() methods return values or pointers to data structures internal to LAMMPS.

For extract_global() see the src/library.cpp file for the list of valid names. New names could easily be added. A double or integer is returned. You need to specify the appropriate data type via the type argument.

For extract_atom(), a pointer to internal LAMMPS atom−based data is returned, which you can use via normal Python subscripting. See the extract() method in the src/atom.cpp file for a list of valid names. Again, new names could easily be added. A pointer to a vector of doubles or integers, or a pointer to an array of doubles (double **) is returned. You need to specify the appropriate data type via the type argument.

For extract_compute() and extract_fix(), the global, per−atom, or local data calulated by the compute or fix can be accessed. What is returned depends on whether the compute or fix calculates a scalar or vector or array. For a

scalar, a single double value is returned. If the compute or fix calculates a vector or array, a pointer to the internal LAMMPS data is returned, which you can use via normal Python subscripting. The one exception is that for a fix that calculates a global vector or array, a single double value from the vector or array is returned, indexed by I (vector) or I and J (array). I,J are zero−based indices. The I,J arguments can be left out if not needed. See this section of the manual for a discussion of global, per−atom, and local data, and of scalar, vector, and array data types. See the doc pages for individual computes and fixes for a description of what they calculate and store.

For extract_variable(), an equal−style or atom−style variable is evaluated and its result returned.

For equal−style variables a single double value is returned and the group argument is ignored. For atom−style variables, a vector of doubles is returned, one value per atom, which you can use via normal Python subscripting. The values will be zero for atoms not in the specified group.

The get_natoms() method returns the total number of atoms in the simulation, as an int. Note that extract_global("natoms") returns the same value, but as a double, which is the way LAMMPS stores it to allow for systems with more atoms than can be stored in an int (> 2 billion).

The get_coords() method returns an ctypes vector of doubles of length 3*natoms, for the coordinates of all the atoms in the simulation, ordered by x,y,z and then by atom ID (see code for put_coords() below). The array can be used via normal Python subscripting. If atom IDs are not consecutively ordered within LAMMPS, a None is returned as indication of an error.

Note that the data structure get_coords() returns is different from the data structure returned by extract_atom("x") in four ways. (1) Get_coords() returns a vector which you index as x[i]; extract_atom() returns an array which you index as x[i][j]. (2) Get_coords() orders the atoms by atom ID while extract_atom() does not. (3) Get_coords() returns a list of all atoms in the simulation; extract_atoms() returns just the atoms local to each processor. (4) Finally, the get_coords() data structure is a copy of the atom coords stored internally in LAMMPS, whereas extract_atom returns an array that points directly to the internal data. This means you can change values inside LAMMPS from Python by assigning a new values to the extract_atom() array. To do this with the get_atoms() vector, you need to change values in the vector, then invoke the put_coords() method.

The put_coords() method takes a vector of coordinates for all atoms in the simulation, assumed to be ordered by x,y,z and then by atom ID, and uses the values to overwrite the corresponding coordinates for each atom inside LAMMPS. This requires LAMMPS to have its "map" option enabled; see the atom_modify command for details. If it is not or if atom IDs are not consecutively ordered, no coordinates are reset,

The array of coordinates passed to put_coords() must be a ctypes vector of doubles, allocated and initialized something like this:

```
from ctypes import *
natoms = lmp.get_atoms()
n3 = 3*natoms
x = (c_double*n3)()
x0 = x coord of atom with ID 1
x1 = y coord of atom with ID 1
x2 = z coord of atom with ID 1
x3 = x coord of atom with ID 2
...
xn3-1 = z coord of atom with ID natoms
lmp.put_coords(x)
```

Alternatively, you can just change values in the vector returned by get_coords(), since it is a ctypes vector of doubles.

As noted above, these Python class methods correspond one−to−one with the functions in the LAMMPS library interface in src/library.cpp and library.h. This means you can extend the Python wrapper via the following steps:

- Add a new interface function to src/library.cpp and src/library.h.
- Verify the new function is syntactically correct by building LAMMPS as a library − see this section of the manual.
- Add a wrapper method in the Python LAMMPS module to python/lammps.py for this interface function.
- Rebuild the Python wrapper via python/setup_serial.py or python/setup.py.
- You should now be able to invoke the new interface function from a Python script. Isn't ctypes amazing?

11.7 Example Python scripts that use LAMMPS

These are the Python scripts included as demos in the python/examples directory of the LAMMPS distribution, to illustrate the kinds of things that are possible when Python wraps LAMMPS. If you create your own scripts, send them to us and we can include them in the LAMMPS distribution.

trivial.py	read/run a LAMMPS input script thru Python
demo.py	invoke various LAMMPS library interface routines
simple.py	mimic operation of couple/simple/simple.cpp in Python
gui.py	GUI go/stop/temperature−slider to control LAMMPS
plot.py	real−time temeperature plot with GnuPlot via Pizza.py
viz_tool.py	real−time viz via some viz package
vizplotgui_tool.py	combination of viz_tool.py and plot.py and gui.py

---

For the viz_tool.py and vizplotgui_tool.py commands, replace "tool" with "gl" or "atomeye" or "pymol" or "vmd", depending on what visualization package you have installed.

Note that for GL, you need to be able to run the Pizza.py GL tool, which is included in the pizza sub−directory. See the Pizza.py doc pages for more info:

Note that for AtomEye, you need version 3, and there is a line in the scripts that specifies the path and name of the executable. See the AtomEye WWW pages here or here for more details:

```
http://mt.seas.upenn.edu/Archive/Graphics/A
http://mt.seas.upenn.edu/Archive/Graphics/A3/A3.html
```

The latter link is to AtomEye 3 which has the scriping capability needed by these Python scripts.

Note that for PyMol, you need to have built and installed the open−source version of PyMol in your Python, so that you can import it from a Python script. See the PyMol WWW pages here or here for more details:

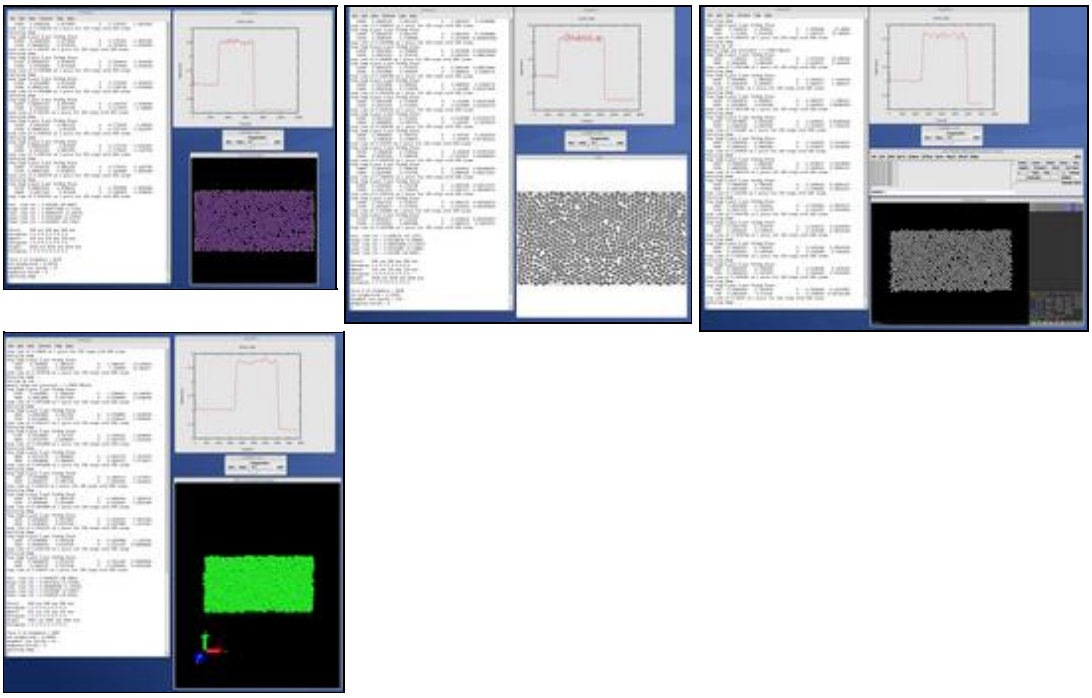
```
http://www.pymol.org
http://sourceforge.net/scm/?type=svn
```

The latter link is to the open−source version.

Note that for VMD, you need a fairly current version (1.8.7 works for me) and there are some lines in the pizza/vmd.py script for 4 PIZZA variables that have to match the VMD installation on your system.

See the python/README file for instructions on how to run them and the source code for individual scripts for comments about what they do.

Here are screenshots of the vizplotgui_tool.py script in action for different visualization package options. Click to see larger images:



12. Errors

This section describes the various kinds of errors you can encounter when using LAMMPS.

12.1 Common problems
12.2 Reporting bugs
12.3 Error &warning messages

12.1 Common problems

If two LAMMPS runs do not produce the same answer on different machines or different numbers of processors, this is typically not a bug. In theory you should get identical answers on any number of processors and on any machine. In practice, numerical round−off can cause slight differences and eventual divergence of molecular dynamics phase space trajectories within a few 100s or few 1000s of timesteps. However, the statistical properties of the two runs (e.g. average energy or temperature) should still be the same.

If the velocity command is used to set initial atom velocities, a particular atom can be assigned a different velocity when the problem is run on a different number of processors or on different machines. If this happens, the phase space trajectories of the two simulations will rapidly diverge. See the discussion of the *loop* option in the velocity command for details and options that avoid this issue.

Similarly, the create_atoms command generates a lattice of atoms. For the same physical system, the ordering and numbering of atoms by atom ID may be different depending on the number of processors.

Some commands use random number generators which may be setup to produce different random number streams on each processor and hence will produce different effects when run on different numbers of processors. A commonly−used example is the fix langevin command for thermostatting.

A LAMMPS simulation typically has two stages, setup and run. Most LAMMPS errors are detected at setup time; others like a bond stretching too far may not occur until the middle of a run.

LAMMPS tries to flag errors and print informative error messages so you can fix the problem. Of course, LAMMPS cannot figure out your physics or numerical mistakes, like choosing too big a timestep, specifying erroneous force field coefficients, or putting 2 atoms on top of each other! If you run into errors that LAMMPS doesn't catch that you think it should flag, please send an email to the developers.

If you get an error message about an invalid command in your input script, you can determine what command is causing the problem by looking in the log.lammps file or using the echo command to see it on the screen. For a given command, LAMMPS expects certain arguments in a specified order. If you mess this up, LAMMPS will often flag the error, but it may read a bogus argument and assign a value that is valid, but not what you wanted. E.g. trying to read the string "abc" as an integer value and assigning the associated variable a value of 0.

Generally, LAMMPS will print a message to the screen and logfile and exit gracefully when it encounters a fatal error. Sometimes it will print a WARNING to the screen and logfile and continue on; you can decide if the WARNING is important or not. A WARNING message that is generated in the middle of a run is only printed to the screen, not to the logfile, to avoid cluttering up thermodynamic output. If LAMMPS crashes or hangs without spitting out an error message first then it could be a bug (see this section) or one of the following cases:

LAMMPS runs in the available memory a processor allows to be allocated. Most reasonable MD runs are compute limited, not memory limited, so this shouldn't be a bottleneck on most platforms. Almost all large memory allocations in the code are done via C−style malloc's which will generate an error message if you run out of memory. Smaller chunks of memory are allocated via C++ "new" statements. If you are unlucky you could run out of memory just when one of these small requests is made, in which case the code will crash or hang (in parallel), since LAMMPS doesn't trap on those errors.

Illegal arithmetic can cause LAMMPS to run slow or crash. This is typically due to invalid physics and numerics that your simulation is computing. If you see wild thermodynamic values or NaN values in your LAMMPS output, something is wrong with your simulation. If you suspect this is happening, it is a good idea to print out thermodynamic info frequently (e.g. every timestep) via the thermo so you can monitor what is happening. Visualizing the atom movement is also a good idea to insure your model is behaving as you expect.

In parallel, one way LAMMPS can hang is due to how different MPI implementations handle buffering of messages. If the code hangs without an error message, it may be that you need to specify an MPI setting or two (usually via an environment variable) to enable buffering or boost the sizes of messages that can be buffered.

12.2 Reporting bugs

If you are confident that you have found a bug in LAMMPS, follow these steps.

Check the New features and bug fixes section of the LAMMPS WWW site to see if the bug has already been reported or fixed or the Unfixed bug to see if a fix is pending.

Check the mailing list to see if it has been discussed before.

If not, send an email to the mailing list describing the problem with any ideas you have as to what is causing it or where in the code the problem might be. The developers will ask for more info if needed, such as an input script or data files.

The most useful thing you can do to help us fix the bug is to isolate the problem. Run it on the smallest number of atoms and fewest number of processors and with the simplest input script that reproduces the bug and try to identify what command or combination of commands is causing the problem.

As a last resort, you can send an email directly to the developers.

12.3 Error &warning messages

These are two alphabetic lists of the ERROR and WARNING messages LAMMPS prints out and the reason why. If the explanation here is not sufficient, the documentation for the offending command may help. Grepping the source files for the text of the error message and staring at the source code and comments is also not a bad idea! Note that sometimes the same message can be printed from multiple places in the code.

Also note that error messages from user−contributed packages are not listed here. Is such an error occurs and is not self−explanatory, you'll need to look in the source code or contact the author of the package.

Errors:

1−3 bond count is inconsistent

An inconsistency was detected when computing the number of 1−3 neighbors for each atom. This likely means something is wrong with the bond topologies you have defined.

1−4 bond count is inconsistent
: An inconsistency was detected when computing the number of 1−4 neighbors for each atom. This likely means something is wrong with the bond topologies you have defined.

Accelerated style in input script but no fix gpu
: GPU acceleration requires fix gpu in the input script.

All angle coeffs are not set
: All angle coefficients must be set in the data file or by the angle_coeff command before running a simulation.

All bond coeffs are not set
: All bond coefficients must be set in the data file or by the bond_coeff command before running a simulation.

All dihedral coeffs are not set
: All dihedral coefficients must be set in the data file or by the dihedral_coeff command before running a simulation.

All dipole moments are not set
: For atom styles that define dipole moments for each atom type, all moments must be set in the data file or by the dipole command before running a simulation.

All improper coeffs are not set
: All improper coefficients must be set in the data file or by the improper_coeff command before running a simulation.

All masses are not set
: For atom styles that define masses for each atom type, all masses must be set in the data file or by the mass command before running a simulation. They must also be set before using the velocity command.

All pair coeffs are not set
: All pair coefficients must be set in the data file or by the pair_coeff command before running a simulation.

All shapes are not set
: All atom types must have a shape setting, even if the particles are spherical.

All universe/uloop variables must have same # of values
: Self−explanatory.

All variables in next command must be same style
: Self−explanatory.

Angle atom missing in delete_bonds
: The delete_bonds command cannot find one or more atoms in a particular angle on a particular processor. The pairwise cutoff is too short or the atoms are too far apart to make a valid angle.

Angle atom missing in set command
: The set command cannot find one or more atoms in a particular angle on a particular processor. The pairwise cutoff is too short or the atoms are too far apart to make a valid angle.

Angle atoms %d %d %d missing on proc %d at step
: One or more of 3 atoms needed to compute a particular angle are missing on this processor. Typically this is because the pairwise cutoff is set too short or the angle has blown apart and an atom is too far away.

Angle coeff for hybrid has invalid style
: Angle style hybrid uses another angle style as one of its coefficients. The angle style used in the angle_coeff command or read from a restart file is not recognized.

Angle coeffs are not set
: No angle coefficients have been assigned in the data file or via the angle_coeff command.

Angle potential must be defined for SHAKE
: When shaking angles, an angle_style potential must be used.

Angle style hybrid cannot have hybrid as an argument
: Self−explanatory.

Angle style hybrid cannot have none as an argument
: Self−explanatory.

Angle style hybrid cannot use same pair style twice
: Self−explanatory.

Angle table must range from 0 to 180 degrees
: Self−explanatory.

Angle table parameters did not set N
: List of angle table parameters must include N setting.

Angle_coeff command before angle_style is defined
: Coefficients cannot be set in the data file or via the angle_coeff command until an angle_style has been assigned.

Angle_coeff command before simulation box is defined
: The angle_coeff command cannot be used before a read_data, read_restart, or create_box command.

Angle_coeff command when no angles allowed
: The chosen atom style does not allow for angles to be defined.

Angle_style command when no angles allowed
: The chosen atom style does not allow for angles to be defined.

Angles assigned incorrectly
: Angles read in from the data file were not assigned correctly to atoms. This means there is something invalid about the topology definitions.

Angles defined but no angle types
: The data file header lists angles but no angle types.

Another input script is already being processed
: Cannot attempt to open a 2nd input script, when the original file is still being processed.

Arccos of invalid value in variable formula
: Argument of arccos() must be between −1 and 1.

Arcsin of invalid value in variable formula
: Argument of arcsin() must be between −1 and 1.

Atom IDs must be consecutive for velocity create loop all
: Self−explanatory.

Atom count changed in fix neb
: This is not allowed in a NEB calculation.

Atom count is inconsistent, cannot write restart file
: Sum of atoms across processors does not equal initial total count. This is probably because you have lost some atoms.

Atom in too many rigid bodies − boost MAXBODY
: Fix poems has a parameter MAXBODY (in fix_poems.cpp) which determines the maximum number of rigid bodies a single atom can belong to (i.e. a multibody joint). The bodies you have defined exceed this limit.

Atom sort did not operate correctly
: This is an internal LAMMPS error. Please report it to the developers.

Atom sorting has bin size = 0.0
: The neighbor cutoff is being used as the bin size, but it is zero. Thus you must explicitly list a bin size in the atom_modify sort command or turn off sorting.

Atom style hybrid cannot have hybrid as an argument
: Self−explanatory.

Atom style hybrid cannot use same atom style twice
: Self−explanatory.

Atom vector in equal−style variable formula
: Atom vectors generate one value per atom which is not allowed in an equal−style variable.

Atom−style variable in equal−style variable formula
: Atom−style variables generate one value per atom which is not allowed in an equal−style variable.

Atom_modify map command after simulation box is defined
: The atom_modify map command cannot be used after a read_data, read_restart, or create_box command.

Atom_modify sort and first options cannot be used together
Self−explanatory.
Atom_style command after simulation box is defined
The atom_style command cannot be used after a read_data, read_restart, or create_box command.
Attempt to pop empty stack in fix box/relax
Internal LAMMPS error. Please report it to the developers.
Attempt to push beyond stack limit in fix box/relax
Internal LAMMPS error. Please report it to the developers.
Attempting to rescale a 0.0 temperature
Cannot rescale a temperature that is already 0.0.
Bad FENE bond
Two atoms in a FENE bond have become so far apart that the bond cannot be computed.
Bad TIP4P angle type for PPPM/TIP4P
Specified angle type is not valid.
Bad TIP4P bond type for PPPM/TIP4P
Specified bond type is not valid.
Bad grid of processors
The 3d grid of processors defined by the processors command does not match the number of processors LAMMPS is being run on.
Bad kspace_modify slab parameter
Kspace_modify value for the slab/volume keyword must be >= 2.0.
Bad principal moments
Fix rigid did not compute the principal moments of inertia of a rigid group of atoms correctly.
Bias compute does not calculate a velocity bias
The specified compute must compute a bias for temperature.
Bias compute does not calculate temperature
The specified compute must compute temperature.
Bias compute group does not match compute group
The specified compute must operate on the same group as the parent compute.
Big particle in fix srd cannot be point particle
Big particles must be extended spheriods or ellipsoids.
Bigint setting in lmptype.h is invalid
Size of bigint is less than size of tagint.
Bigint setting in lmptype.h is not compatible
Bigint stored in restart file is not consistent with LAMMPS version you are running.
Bitmapped lookup tables require int/float be same size
Cannot use pair tables on this machine, because of word sizes. Use the pair_modify command with table 0 instead.
Bitmapped table in file does not match requested table
Setting for bitmapped table in pair_coeff command must match table in file exactly.
Bitmapped table is incorrect length in table file
Number of table entries is not a correct power of 2.
Bond and angle potentials must be defined for TIP4P
Cannot use TIP4P pair potential unless bond and angle potentials are defined.
Bond atom missing in delete_bonds
The delete_bonds command cannot find one or more atoms in a particular bond on a particular processor. The pairwise cutoff is too short or the atoms are too far apart to make a valid bond.
Bond atom missing in set command
The set command cannot find one or more atoms in a particular bond on a particular processor. The pairwise cutoff is too short or the atoms are too far apart to make a valid bond.
Bond atoms %d %d missing on proc %d at step
One or both of 2 atoms needed to compute a particular bond are missing on this processor. Typically this

is because the pairwise cutoff is set too short or the bond has blown apart and an atom is too far away.

Bond coeff for hybrid has invalid style

Bond style hybrid uses another bond style as one of its coefficients. The bond style used in the bond_coeff command or read from a restart file is not recognized.

Bond coeffs are not set

No bond coefficients have been assigned in the data file or via the bond_coeff command.

Bond potential must be defined for SHAKE

Cannot use fix shake unless bond potential is defined.

Bond style hybrid cannot have hybrid as an argument

Self−explanatory.

Bond style hybrid cannot have none as an argument

Self−explanatory.

Bond style hybrid cannot use same pair style twice

Self−explanatory.

Bond style quartic cannot be used with 3,4−body interactions

No angle, dihedral, or improper styles can be defined when using bond style quartic.

Bond style quartic requires special_bonds = 1,1,1

This is a restriction of the current bond quartic implementation.

Bond table parameters did not set N

List of bond table parameters must include N setting.

Bond table values are not increasing

The values in the tabulated file must be monotonically increasing.

Bond_coeff command before bond_style is defined

Coefficients cannot be set in the data file or via the bond_coeff command until an bond_style has been assigned.

Bond_coeff command before simulation box is defined

The bond_coeff command cannot be used before a read_data, read_restart, or create_box command.

Bond_coeff command when no bonds allowed

The chosen atom style does not allow for bonds to be defined.

Bond_style command when no bonds allowed

The chosen atom style does not allow for bonds to be defined.

Bonds assigned incorrectly

Bonds read in from the data file were not assigned correctly to atoms. This means there is something invalid about the topology definitions.

Bonds defined but no bond types

The data file header lists bonds but no bond types.

Both sides of boundary must be periodic

Cannot specify a boundary as periodic only on the lo or hi side. Must be periodic on both sides.

Boundary command after simulation box is defined

The boundary command cannot be used after a read_data, read_restart, or create_box command.

Box bounds are invalid

The box boundaries specified in the read_data file are invalid. The lo value must be less than the hi value for all 3 dimensions.

Can not specify Pxy/Pxz/Pyz in fix box/relax with non−triclinic box

Only triclinic boxes can be used with off−diagonal pressure components. See the region prism command for details.

Can not specify Pxy/Pxz/Pyz in fix nvt/npt/nph with non−triclinic box

Only triclinic boxes can be used with off−diagonal pressure components. See the region prism command for details.

Can only use NEB with 1−processor replicas

This is current restriction for NEB as implemented in LAMMPS.

Can only use TAD with 1−processor replicas for NEB

This is current restriction for NEB as implemented in LAMMPS.

Cannot (yet) use PPPM with triclinic box

This feature is not yet supported.

Cannot add atoms to fix move variable

Atoms can not be added afterwards to this fix option.

Cannot change box to orthogonal when tilt is non−zero

Self−explanatory

Cannot change box with certain fixes defined

The change_box command cannot be used when fix ave/spatial or fix/deform are defined .

Cannot change box with dumps defined

Self−explanatory.

Cannot change dump_modify every for dump dcd

The frequency of writing dump dcd snapshots cannot be changed.

Cannot change dump_modify every for dump xtc

The frequency of writing dump xtc snapshots cannot be changed.

Cannot change timestep once fix srd is setup

This is because various SRD properties depend on the timestep size.

Cannot change timestep with fix pour

This fix pre−computes some values based on the timestep, so it cannot be changed during a simulation run.

Cannot compute PPPM G

LAMMPS failed to compute a valid approximation for the PPPM g_ewald factor that partitions the computation between real space and k−space.

Cannot create an atom map unless atoms have IDs

The simulation requires a mapping from global atom IDs to local atoms, but the atoms that have been defined have no IDs.

Cannot create atoms with undefined lattice

Must use the lattice command before using the create_atoms command.

Cannot create/grow a vector/array of pointers for %s

LAMMPS code is making an illegal call to the templated memory allocaters, to create a vector or array of pointers.

Cannot create_atoms after reading restart file with per−atom info

The per−atom info was stored to be used when by a fix that you may re−define. If you add atoms before re−defining the fix, then there will not be a correct amount of per−atom info.

Cannot create_box after simulation box is defined

The create_box command cannot be used after a read_data, read_restart, or create_box command.

Cannot currently use pair reax with pair hybrid

This is not yet supported.

Cannot delete group all

Self−explanatory.

Cannot delete group currently used by a compute

Self−explanatory.

Cannot delete group currently used by a dump

Self−explanatory.

Cannot delete group currently used by a fix

Self−explanatory.

Cannot delete group currently used by atom_modify first

Self−explanatory.

Cannot displace_atoms after reading restart file with per−atom info

This is because the restart file info cannot be migrated with the atoms. You can get around this by performing a 0−timestep run which will assign the restart file info to actual atoms.

Cannot displace_box after reading restart file with per−atom info

This is because the restart file info cannot be migrated with the atoms. You can get around this by performing a 0−timestep run which will assign the restart file info to actual atoms.

Cannot displace_box on a non−periodic boundary

Self−explanatory.

Cannot dump sort on atom IDs with no atom IDs defined

Self−explanatory.

Cannot evaporate atoms in atom_modify first group

This is a restriction due to the way atoms are organized in a list to enable the atom_modify first command.

Cannot find delete_bonds group ID

Group ID used in the delete_bonds command does not exist.

Cannot have both pair_modify shift and tail set to yes

These 2 options are contradictory.

Cannot open AIREBO potential file %s

The specified AIREBO potential file cannot be opened. Check that the path and name are correct.

Cannot open COMB potential file %s

The specified COMB potential file cannot be opened. Check that the path and name are correct.

Cannot open EAM potential file %s

The specified EAM potential file cannot be opened. Check that the path and name are correct.

Cannot open EIM potential file %s

The specified EIM potential file cannot be opened. Check that the path and name are correct.

Cannot open MEAM potential file %s

The specified MEAM potential file cannot be opened. Check that the path and name are correct.

Cannot open Stillinger−Weber potential file %s

The specified SW potential file cannot be opened. Check that the path and name are correct.

Cannot open Tersoff potential file %s

The specified Tersoff potential file cannot be opened. Check that the path and name are correct.

Cannot open dir to search for restart file

Using a "*" in the name of the restart file will open the current directory to search for matching file names.

Cannot open dump file

The output file for the dump command cannot be opened. Check that the path and name are correct.

Cannot open file %s

The specified file cannot be opened. Check that the path and name are correct.

Cannot open fix ave/correlate file %s

The specified file cannot be opened. Check that the path and name are correct.

Cannot open fix ave/histo file %s

The specified file cannot be opened. Check that the path and name are correct.

Cannot open fix ave/spatial file %s

The specified file cannot be opened. Check that the path and name are correct.

Cannot open fix ave/time file %s

The specified file cannot be opened. Check that the path and name are correct.

Cannot open fix poems file %s

The specified file cannot be opened. Check that the path and name are correct.

Cannot open fix print file %s

The output file generated by the fix print command cannot be opened

Cannot open fix qeq/comb file %s

The output file for the fix qeq/combs command cannot be opened. Check that the path and name are correct.

Cannot open fix reax/bonds file %s

The output file for the fix reax/bonds command cannot be opened. Check that the path and name are correct.

Cannot open fix tmd file %s
: The output file for the fix tmd command cannot be opened. Check that the path and name are correct.

Cannot open fix ttm file %s
: The output file for the fix ttm command cannot be opened. Check that the path and name are correct.

Cannot open gzipped file
: LAMMPS is attempting to open a gzipped version of the specified file but was unsuccessful. Check that the path and name are correct.

Cannot open input script %s
: Self−explanatory.

Cannot open log.lammps
: The default LAMMPS log file cannot be opened. Check that the directory you are running in allows for files to be created.

Cannot open logfile %s
: The LAMMPS log file specified in the input script cannot be opened. Check that the path and name are correct.

Cannot open logfile
: The LAMMPS log file named in a command−line argument cannot be opened. Check that the path and name are correct.

Cannot open pair_write file
: The specified output file for pair energies and forces cannot be opened. Check that the path and name are correct.

Cannot open restart file %s
: Self−explanatory.

Cannot open screen file
: The screen file specified as a command−line argument cannot be opened. Check that the directory you are running in allows for files to be created.

Cannot open universe log file
: For a multi−partition run, the master log file cannot be opened. Check that the directory you are running in allows for files to be created.

Cannot open universe screen file
: For a multi−partition run, the master screen file cannot be opened. Check that the directory you are running in allows for files to be created.

Cannot read_data after simulation box is defined
: The read_data command cannot be used after a read_data, read_restart, or create_box command.

Cannot read_restart after simulation box is defined
: The read_restart command cannot be used after a read_data, read_restart, or create_box command.

Cannot redefine variable as a different style
: An equal−style variable can be re−defined but only if it was originally an equal−style variable.

Cannot replicate 2d simulation in z dimension
: The replicate command cannot replicate a 2d simulation in the z dimension.

Cannot replicate with fixes that store atom quantities
: Either fixes are defined that create and store atom−based vectors or a restart file was read which included atom−based vectors for fixes. The replicate command cannot duplicate that information for new atoms. You should use the replicate command before fixes are applied to the system.

Cannot reset timestep with a dynamic region defined
: Dynamic regions (see the region command) have a time dependence. Thus you cannot change the timestep when one or more of these are defined.

Cannot reset timestep with a time−dependent fix defined
: You cannot reset the timestep when a fix that keeps track of elapsed time is in place.

Cannot reset timestep with dump file already written to
: Changing the timestep will confuse when a dump file is written. Use the undump command, then restart the dump file.

Cannot reset timestep with restart file already written
: Changing the timestep will confuse when a restart file is written. Use the "restart 0" command to turn off restarts, then start them again.

Cannot restart fix rigid/nvt with different # of chains
: This is because the restart file contains per−chain info.

Cannot run 2d simulation with nonperiodic Z dimension
: Use the boundary command to make the z dimension periodic in order to run a 2d simulation.

Cannot set both respa pair and inner/middle/outer
: In the rRESPA integrator, you must compute pairwise potentials either all together (pair), or in pieces (inner/middle/outer). You can't do both.

Cannot set dipole for this atom style
: This atom style does not support dipole settings for each atom type.

Cannot set dump_modify flush for dump xtc
: Self−explanatory.

Cannot set mass for this atom style
: This atom style does not support mass settings for each atom type. Instead they are defined on a per−atom basis in the data file.

Cannot set non−zero image flag for non−periodic dimension
: Self−explanatory.

Cannot set non−zero z velocity for 2d simulation
: Self−explanatory.

Cannot set respa middle without inner/outer
: In the rRESPA integrator, you must define both a inner and outer setting in order to use a middle setting.

Cannot set shape for this atom style
: The atom style does not support this setting.

Cannot set this attribute for this atom style
: The attribute being set does not exist for the defined atom style.

Cannot set variable z velocity for 2d simulation
: Self−explanatory.

Cannot skew triclinic box in z for 2d simulation
: Self−explanatory.

Cannot use Ewald with 2d simulation
: The kspace style ewald cannot be used in 2d simulations. You can use 2d Ewald in a 3d simulation; see the kspace_modify command.

Cannot use Ewald with triclinic box
: This feature is not yet supported.

Cannot use NEB unless atom map exists
: Use the atom_modify command to create an atom map.

Cannot use NEB with a single replica
: Self−explanatory.

Cannot use NEB with atom_modify sort enabled
: This is current restriction for NEB implemented in LAMMPS.

Cannot use PPPM with 2d simulation
: The kspace style pppm cannot be used in 2d simulations. You can use 2d PPPM in a 3d simulation; see the kspace_modify command.

Cannot use PRD with a time−dependent fix defined
: PRD alters the timestep in ways that will mess up these fixes.

Cannot use PRD with a time−dependent region defined
: PRD alters the timestep in ways that will mess up these regions.

Cannot use PRD with atom_modify sort enabled
: This is a current restriction of PRD. You must turn off sorting, which is enabled by default, via the atom_modify command.

Cannot use PRD with multi−processor replicas unless atom map exists
Use the atom_modify command to create an atom map.
Cannot use TAD unless atom map exists for NEB
See atom_modify map command to set this.
Cannot use TAD with a single replica for NEB
NEB requires multiple replicas.
Cannot use TAD with atom_modify sort enabled for NEB
This is a current restriction of NEB.
Cannot use a damped dynamics min style with fix box/relax
This is a current restriction in LAMMPS. Use another minimizer style.
Cannot use a damped dynamics min style with per−atom DOF
This is a current restriction in LAMMPS. Use another minimizer style.
Cannot use compute cluster/atom unless atoms have IDs
Atom IDs are used to identify clusters.
Cannot use cwiggle in variable formula between runs
This is a function of elapsed time.
Cannot use delete_atoms unless atoms have IDs
Your atoms do not have IDs, so the delete_atoms command cannot be used.
Cannot use delete_bonds with non−molecular system
Your choice of atom style does not have bonds.
Cannot use fix TMD unless atom map exists
Using this fix requires the ability to lookup an atom index, which is provided by an atom map. An atom map does not exist (by default) for non−molecular problems. Using the atom_modify map command will force an atom map to be created.
Cannot use fix ave/spatial z for 2 dimensional model
Self−explanatory.
Cannot use fix bond/break with non−molecular systems
Self−explanatory.
Cannot use fix bond/create with non−molecular systems
Self−explanatory.
Cannot use fix box/relax on a 2nd non−periodic dimension
When specifying an off−diagonal pressure component, the 2nd of the two dimensions must be periodic. E.g. if the xy component is specified, then the y dimension must be periodic.
Cannot use fix box/relax on a non−periodic dimension
When specifying a diagonal pressure component, the dimension must be periodic.
Cannot use fix deform on a 2nd non−periodic boundary
When specifying a tilt factor change, the 2nd of the two dimensions must be periodic. E.g. if the xy tilt is specified, then the y dimension must be periodic.
Cannot use fix deform on a non−periodic boundary
When specifying a change is a box dimension, the dimension must be periodic.
Cannot use fix deform trate on a box with zero tilt
The trate style alters the current strain.
Cannot use fix enforce2d with 3d simulation
Self−explanatory.
Cannot use fix msst without per−type mass defined
Self−explanatory.
Cannot use fix npt and fix deform on same component of stress tensor
This would be changing the same box dimension twice.
Cannot use fix nvt/npt/nph on a 2nd non−periodic dimension
When specifying an off−diagonal pressure component, the 2nd of the two dimensions must be periodic. E.g. if the xy component is specified, then the y dimension must be periodic.
Cannot use fix nvt/npt/nph on a non−periodic dimension

When specifying a diagonal pressure component, the dimension must be periodic.

Cannot use fix pour with triclinic box

This feature is not yet supported.

Cannot use fix press/berendsen and fix deform on same component of stress tensor

These commands both change the box size/shape, so you cannot use both together.

Cannot use fix press/berendsen on a non−periodic dimension

Self−explanatory.

Cannot use fix press/berendsen with triclinic box

Self−explanatory.

Cannot use fix reax/bonds without pair_style reax

Self−explantory.

Cannot use fix shake with non−molecular system

Your choice of atom style does not have bonds.

Cannot use fix ttm with 2d simulation

This is a current restriction of this fix due to the grid it creates.

Cannot use fix ttm with triclinic box

This is a current restriction of this fix due to the grid it creates.

Cannot use fix wall in periodic dimension

Self−explanatory.

Cannot use fix wall zlo/zhi for a 2d simulation

Self−explanatory.

Cannot use fix wall/reflect in periodic dimension

Self−explanatory.

Cannot use fix wall/reflect zlo/zhi for a 2d simulation

Self−explanatory.

Cannot use fix wall/srd in periodic dimension

Self−explanatory.

Cannot use fix wall/srd more than once

Nor is their a need to since multiple walls can be specified in one command.

Cannot use fix wall/srd without fix srd

Self−explanatory.

Cannot use fix wall/srd zlo/zhi for a 2d simulation

Self−explanatory.

Cannot use force/neigh with triclinic box

This is a current limitation of the GPU implementation in LAMMPS.

Cannot use kspace solver on system with no charge

No atoms in system have a non−zero charge.

Cannot use neighbor bins − box size << cutoff

Too many neighbor bins will be created. This typically happens when the simulation box is very small in some dimension, compared to the neighbor cutoff. Use the "nsq" style instead of "bin" style.

Cannot use newton pair with GPU CHARMM pair style

See the newton command to change the setting.

Cannot use newton pair with GPU Gay−Berne pair style

See the newton command to change the setting.

Cannot use newton pair with GPU LJ pair style

See the newton command to change the setting.

Cannot use newton pair with GPU LJ96 pair style

See the newton command to change the setting.

Cannot use non−zero forces in an energy minimization

Fix setforce cannot be used in this manner. Use fix addforce instead.

Cannot use nonperiodic boundares with fix ttm

This fix requires a fully periodic simulation box.

Cannot use nonperiodic boundaries with Ewald
: For kspace style ewald, all 3 dimensions must have periodic boundaries unless you use the kspace_modify command to define a 2d slab with a non−periodic z dimension.

Cannot use nonperiodic boundaries with PPPM
: For kspace style pppm, all 3 dimensions must have periodic boundaries unless you use the kspace_modify command to define a 2d slab with a non−periodic z dimension.

Cannot use pair hybrid with GPU neighbor builds
: See documentation for fix gpu.

Cannot use pair tail corrections with 2d simulations
: The correction factors are only currently defined for 3d systems.

Cannot use ramp in variable formula between runs
: This is because the ramp() function is time dependent.

Cannot use region INF or EDGE when box does not exist
: Regions that extend to the box boundaries can only be used after the create_box command has been used.

Cannot use set atom with no atom IDs defined
: Atom IDs are not defined, so they cannot be used to identify an atom.

Cannot use swiggle in variable formula between runs
: This is a function of elapsed time.

Cannot use variable energy with constant force in fix addforce
: This is because for constant force, LAMMPS can compute the change in energy directly.

Cannot use variable every setting for dump dcd
: The format of DCD dump files requires snapshots be output at a constant frequency.

Cannot use variable every setting for dump xtc
: The format of this file requires snapshots at regular intervals.

Cannot use vdisplace in variable formula between runs
: This is a function of elapsed time.

Cannot use velocity create loop all unless atoms have IDs
: Atoms in the simulation to do not have IDs, so this style of velocity creation cannot be performed.

Cannot use wall in periodic dimension
: Self−explanatory.

Cannot wiggle and shear fix wall/gran
: Cannot specify both options at the same time.

Cannot zero momentum of 0 atoms
: The collection of atoms for which momentum is being computed has no atoms.

Change_box command before simulation box is defined
: Self−explanatory.

Change_box operation is invalid
: Cannot change orthogonal box to orthogonal or a triclinic box to triclinic.

Communicate group != atom_modify first group
: Self−explanatory.

Compute ID for compute atom/molecule does not exist
: Self−explanatory.

Compute ID for compute reduce does not exist
: Self−explanatory.

Compute ID for fix ave/atom does not exist
: Self−explanatory.

Compute ID for fix ave/correlate does not exist
: Self−explanatory.

Compute ID for fix ave/histo does not exist
: Self−explanatory.

Compute ID for fix ave/spatial does not exist
: Self−explanatory.

Compute ID for fix ave/time does not exist
Self−explanatory.
Compute ID for fix store/state does not exist
Self−explanatory.
Compute ID must be alphanumeric or underscore characters
Self−explanatory.
Compute angle/local used when angles are not allowed
The atom style does not support angles.
Compute atom/molecule compute array is accessed out−of−range
Self−explanatory.
Compute atom/molecule compute does not calculate a per−atom array
Self−explanatory.
Compute atom/molecule compute does not calculate a per−atom vector
Self−explanatory.
Compute atom/molecule compute does not calculate per−atom values
Self−explanatory.
Compute atom/molecule fix array is accessed out−of−range
Self−explanatory.
Compute atom/molecule fix does not calculate a per−atom array
Self−explanatory.
Compute atom/molecule fix does not calculate a per−atom vector
Self−explanatory.
Compute atom/molecule fix does not calculate per−atom values
Self−explanatory.
Compute atom/molecule requires molecular atom style
Self−explanatory.
Compute atom/molecule variable is not atom−style variable
Self−explanatory.
Compute bond/local used when bonds are not allowed
The atom style does not support bonds.
Compute centro/atom requires a pair style be defined
This is because the computation of the centro−symmetry values uses a pairwise neighbor list.
Compute cluster/atom cutoff is longer than pairwise cutoff
Cannot identify clusters beyond cutoff.
Compute cluster/atom requires a pair style be defined
This is so that the pair style defines a cutoff distance which is used to find clusters.
Compute cna/atom cutoff is longer than pairwise cutoff
Self−explantory.
Compute cna/atom requires a pair style be defined
Self−explantory.
Compute com/molecule requires molecular atom style
Self−explanatory.
Compute coord/atom cutoff is longer than pairwise cutoff
Cannot compute coordination at distances longer than the pair cutoff, since those atoms are not in the neighbor list.
Compute coord/atom requires a pair style be defined
Self−explantory.
Compute damage/atom requires peridynamic potential
Damage is a Peridynamic−specific metric. It requires you to be running a Peridynamics simulation.
Compute dihedral/local used when dihedrals are not allowed
The atom style does not support dihedrals.
Compute does not allow an extra compute or fix to be reset

This is an internal LAMMPS error. Please report it to the developers.

Compute erotate/asphere cannot be used with atom attributes diameter or rmass

These attributes override the shape and mass settings, so cannot be used.

Compute erotate/asphere requires atom attributes angmom, quat, shape

An atom style that defines these attributes must be used.

Compute erotate/asphere requires extended particles

This compute cannot be used with point paritlces.

Compute erotate/sphere requires atom attribute omega

An atom style that defines this attribute must be used.

Compute erotate/sphere requires atom attribute radius or shape

An atom style that defines these attributes must be used.

Compute erotate/sphere requires spherical particle shapes

Self−explanatory.

Compute event/displace has invalid fix event assigned

This is an internal LAMMPS error. Please report it to the developers.

Compute group/group group ID does not exist

Self−explanatory.

Compute gyration/molecule requires molecular atom style

Self−explanatory.

Compute heat/flux compute ID does not compute ke/atom

Self−explanatory.

Compute heat/flux compute ID does not compute pe/atom

Self−explanatory.

Compute heat/flux compute ID does not compute stress/atom

Self−explanatory.

Compute improper/local used when impropers are not allowed

The atom style does not support impropers.

Compute msd/molecule requires molecular atom style

Self−explanatory.

Compute pair must use group all

Pair styles accumlate energy on all atoms.

Compute pe must use group all

Energies computed by potentials (pair, bond, etc) are computed on all atoms.

Compute pressure must use group all

Virial contributions computed by potentials (pair, bond, etc) are computed on all atoms.

Compute pressure temperature ID does not compute temperature

The compute ID assigned to a pressure computation must compute temperature.

Compute property/atom for atom property that isn't allocated

Self−explanatory.

Compute property/local cannot use these inputs together

Only inputs that generate the same number of datums can be used togther. E.g. bond and angle quantities cannot be mixed.

Compute property/local for property that isn't allocated

Self−explanatory.

Compute property/molecule requires molecular atom style

Self−explanatory.

Compute rdf requires a pair style be defined

Self−explanatory.

Compute reduce compute array is accessed out−of−range

Self−explanatory.

Compute reduce compute calculates global values

A compute that calculates peratom or local values is required.

Compute reduce compute does not calculate a local array
Self−explanatory.
Compute reduce compute does not calculate a local vector
Self−explanatory.
Compute reduce compute does not calculate a per−atom array
Self−explanatory.
Compute reduce compute does not calculate a per−atom vector
Self−explanatory.
Compute reduce fix array is accessed out−of−range
Self−explanatory.
Compute reduce fix calculates global values
A fix that calculates peratom or local values is required.
Compute reduce fix does not calculate a local array
Self−explanatory.
Compute reduce fix does not calculate a local vector
Self−explanatory.
Compute reduce fix does not calculate a per−atom array
Self−explanatory.
Compute reduce fix does not calculate a per−atom vector
Self−explanatory.
Compute reduce replace requires min or max mode
Self−explanatory.
Compute reduce variable is not atom−style variable
Self−explanatory.
Compute temp/asphere cannot be used with atom attributes diameter or rmass
These attributes override the shape and mass settings, so cannot be used.
Compute temp/asphere requires atom attributes angmom, quat, shape
An atom style that defines these attributes must be used.
Compute temp/asphere requires extended particles
This compute cannot be used with point paritlces.
Compute temp/partial cannot use vz for 2d systemx
Self−explanatory.
Compute temp/profile cannot bin z for 2d systems
Self−explanatory.
Compute temp/profile cannot use vz for 2d systemx
Self−explanatory.
Compute temp/sphere requires atom attribute omega
An atom style that defines this attribute must be used.
Compute temp/sphere requires atom attribute radius or shape
An atom style that defines these attributes must be used.
Compute temp/sphere requires spherical particle shapes
Self−explanatory.
Compute ti kspace style does not exist
Self−explanatory.
Compute ti pair style does not exist
Self−explanatory.
Compute ti tail when pair style does not compute tail corrections
Self−explanatory.
Compute used in variable between runs is not current
Computes cannot be invoked by a variable in between runs. Thus they must have been evaluated on the last timestep of the previous run in order for their value(s) to be accessed. See the doc page for the variable command for more info.

Compute used in variable thermo keyword between runs is not current
: Some thermo keywords rely on a compute to calculate their value(s). Computes cannot be invoked by a variable in between runs. Thus they must have been evaluated on the last timestep of the previous run in order for their value(s) to be accessed. See the doc page for the variable command for more info.

Computed temperature for fix temp/berendsen cannot be 0.0
: Self−explanatory.

Computed temperature for fix temp/rescale cannot be 0.0
: Cannot rescale the temperature to a new value if the current temperature is 0.0.

Could not count initial bonds in fix bond/create
: Could not find one of the atoms in a bond on this processor.

Could not create 3d FFT plan
: The FFT setup in pppm failed.

Could not create 3d remap plan
: The FFT setup in pppm failed.

Could not find atom_modify first group ID
: Self−explanatory.

Could not find compute ID for PRD
: Self−explanatory.

Could not find compute ID for TAD
: Self−explanatory.

Could not find compute ID for temperature bias
: Self−explanatory.

Could not find compute ID to delete
: Self−explanatory.

Could not find compute displace/atom fix ID
: Self−explanatory.

Could not find compute event/displace fix ID
: Self−explanatory.

Could not find compute group ID
: Self−explanatory.

Could not find compute heat/flux compute ID
: Self−explanatory.

Could not find compute msd fix ID
: Self−explanatory.

Could not find compute pressure temperature ID
: The compute ID for calculating temperature does not exist.

Could not find compute_modify ID
: Self−explanatory.

Could not find delete_atoms group ID
: Group ID used in the delete_atoms command does not exist.

Could not find delete_atoms region ID
: Region ID used in the delete_atoms command does not exist.

Could not find displace_atoms group ID
: Group ID used in the displace_atoms command does not exist.

Could not find displace_box group ID
: Group ID used in the displace_box command does not exist.

Could not find dump cfg compute ID
: Self−explanatory.

Could not find dump cfg fix ID
: Self−explanatory.

Could not find dump cfg variable name
: Self−explanatory.

Could not find dump custom compute ID
: The compute ID needed by dump custom to compute a per−atom quantity does not exist.

Could not find dump custom fix ID
: Self−explanatory.

Could not find dump custom variable name
: Self−explanatory.

Could not find dump group ID
: A group ID used in the dump command does not exist.

Could not find dump local compute ID
: Self−explanatory.

Could not find dump local fix ID
: Self−explanatory.

Could not find dump modify compute ID
: Self−explanatory.

Could not find dump modify fix ID
: Self−explanatory.

Could not find dump modify variable name
: Self−explanatory.

Could not find fix ID to delete
: Self−explanatory.

Could not find fix group ID
: A group ID used in the fix command does not exist.

Could not find fix msst compute ID
: Self−explanatory.

Could not find fix poems group ID
: A group ID used in the fix poems command does not exist.

Could not find fix recenter group ID
: A group ID used in the fix recenter command does not exist.

Could not find fix rigid group ID
: A group ID used in the fix rigid command does not exist.

Could not find fix srd group ID
: Self−explanatory.

Could not find fix_modify ID
: A fix ID used in the fix_modify command does not exist.

Could not find fix_modify pressure ID
: The compute ID for computing pressure does not exist.

Could not find fix_modify temperature ID
: The compute ID for computing temperature does not exist.

Could not find group delete group ID
: Self−explanatory.

Could not find/initialize a specified accelerator device
: Your GPU setup is invalid.

Could not find set group ID
: Group ID specified in set command does not exist.

Could not find thermo compute ID
: Compute ID specified in thermo_style command does not exist.

Could not find thermo custom compute ID
: The compute ID needed by thermo style custom to compute a requested quantity does not exist.

Could not find thermo custom fix ID
: The fix ID needed by thermo style custom to compute a requested quantity does not exist.

Could not find thermo custom variable name
: Self−explanatory.

Could not find thermo fix ID
: Fix ID specified in thermo_style command does not exist.

Could not find thermo_modify pressure ID
: The compute ID needed by thermo style custom to compute pressure does not exist.

Could not find thermo_modify temperature ID
: The compute ID needed by thermo style custom to compute temperature does not exist.

Could not find undump ID
: A dump ID used in the undump command does not exist.

Could not find velocity group ID
: A group ID used in the velocity command does not exist.

Could not find velocity temperature ID
: The compute ID needed by the velocity command to compute temperature does not exist.

Could not grab element entry from EIM potential file
: Self−explanatory

Could not grab global entry from EIM potential file
: Self−explanatory.

Could not grab pair entry from EIM potential file
: Self−explanatory.

Could not set finite−size particle attribute in fix rigid
: The particle has a finite size but its attributes could not be determined.

Coulomb cutoffs of pair hybrid sub−styles do not match
: If using a Kspace solver, all Coulomb cutoffs of long pair styles must be the same.

Cound not find dump_modify ID
: Self−explanatory.

Create_atoms command before simulation box is defined
: The create_atoms command cannot be used before a read_data, read_restart, or create_box command.

Create_atoms region ID does not exist
: A region ID used in the create_atoms command does not exist.

Create_box region ID does not exist
: A region ID used in the create_box command does not exist.

Create_box region does not support a bounding box
: Not all regions represent bounded volumes. You cannot use such a region with the create_box command.

Cyclic loop in joint connections
: Fix poems cannot (yet) work with coupled bodies whose joints connect the bodies in a ring (or cycle).

Degenerate lattice primitive vectors
: Invalid set of 3 lattice vectors for lattice command.

Delete region ID does not exist
: Self−explanatory.

Delete_atoms command before simulation box is defined
: The delete_atoms command cannot be used before a read_data, read_restart, or create_box command.

Delete_atoms cutoff > neighbor cutoff
: Cannot delete atoms further away than a processor knows about.

Delete_atoms requires a pair style be defined
: This is because atom deletion within a cutoff uses a pairwise neighbor list.

Delete_bonds command before simulation box is defined
: The delete_bonds command cannot be used before a read_data, read_restart, or create_box command.

Delete_bonds command with no atoms existing
: No atoms are yet defined so the delete_bonds command cannot be used.

Deposition region extends outside simulation box
: Self−explanatory.

Did not assign all atoms correctly
: Atoms read in from a data file were not assigned correctly to processors. This is likely due to some atom

coordinates being outside a non−periodic simulation box.

Did not find all elements in MEAM library file

The requested elements were not found in the MEAM file.

Did not find fix shake partner info

Could not find bond partners implied by fix shake command. This error can be triggered if the delete_bonds command was used before fix shake, and it removed bonds without resetting the 1−2, 1−3, 1−4 weighting list via the special keyword.

Did not find keyword in table file

Keyword used in pair_coeff command was not found in table file.

Did not set temp for fix rigid/nvt

The temp keyword must be used.

Dihedral atom missing in delete_bonds

The delete_bonds command cannot find one or more atoms in a particular dihedral on a particular processor. The pairwise cutoff is too short or the atoms are too far apart to make a valid dihedral.

Dihedral atom missing in set command

The set command cannot find one or more atoms in a particular dihedral on a particular processor. The pairwise cutoff is too short or the atoms are too far apart to make a valid dihedral.

Dihedral atoms %d %d %d %d missing on proc %d at step

One or more of 4 atoms needed to compute a particular dihedral are missing on this processor. Typically this is because the pairwise cutoff is set too short or the dihedral has blown apart and an atom is too far away.

Dihedral charmm is incompatible with Pair style

Dihedral style charmm must be used with a pair style charmm in order for the 1−4 epsilon/sigma parameters to be defined.

Dihedral coeff for hybrid has invalid style

Dihedral style hybrid uses another dihedral style as one of its coefficients. The dihedral style used in the dihedral_coeff command or read from a restart file is not recognized.

Dihedral coeffs are not set

No dihedral coefficients have been assigned in the data file or via the dihedral_coeff command.

Dihedral style hybrid cannot have hybrid as an argument

Self−explanatory.

Dihedral style hybrid cannot have none as an argument

Self−explanatory.

Dihedral style hybrid cannot use same dihedral style twice

Self−explanatory.

Dihedral_coeff command before dihedral_style is defined

Coefficients cannot be set in the data file or via the dihedral_coeff command until an dihedral_style has been assigned.

Dihedral_coeff command before simulation box is defined

The dihedral_coeff command cannot be used before a read_data, read_restart, or create_box command.

Dihedral_coeff command when no dihedrals allowed

The chosen atom style does not allow for dihedrals to be defined.

Dihedral_style command when no dihedrals allowed

The chosen atom style does not allow for dihedrals to be defined.

Dihedrals assigned incorrectly

Dihedrals read in from the data file were not assigned correctly to atoms. This means there is something invalid about the topology definitions.

Dihedrals defined but no dihedral types

The data file header lists dihedrals but no dihedral types.

Dimension command after simulation box is defined

The dimension command cannot be used after a read_data, read_restart, or create_box command.

Dipole command before simulation box is defined

The dipole command cannot be used before a read_data, read_restart, or create_box command.

Displace_atoms command before simulation box is defined

The displace_atoms command cannot be used before a read_data, read_restart, or create_box command.

Displace_box command before simulation box is defined

Self−explanatory.

Displace_box tilt factors require triclinic box

Cannot use tilt factors unless the simulation box is non−orthogonal.

Distance must be > 0 for compute event/displace

Self−explanatory.

Divide by 0 in influence function of pair peri/lps

This should not normally occur. It is likely a problem with your model.

Divide by 0 in variable formula

Self−explanatory.

Domain too large for neighbor bins

The domain has become extremely large so that neighbor bins cannot be used. Most likely, one or more atoms have been blown out of the simulation box to a great distance.

Double precision is not supported on this accelerator.

In this case, you must compile the GPU library for single precision.

Dump cfg and fix not computed at compatible times

The fix must produce per−atom quantities on timesteps that dump cfg needs them.

Dump cfg arguments must start with 'id type xs ys zs'

This is a requirement of the CFG output format.

Dump cfg requires one snapshot per file

Use the wildcard "*" character in the filename.

Dump custom and fix not computed at compatible times

The fix must produce per−atom quantities on timesteps that dump custom needs them.

Dump custom compute does not calculate per−atom array

Self−explanatory.

Dump custom compute does not calculate per−atom vector

Self−explanatory.

Dump custom compute does not compute per−atom info

Self−explanatory.

Dump custom compute vector is accessed out−of−range

Self−explanatory.

Dump custom fix does not compute per−atom array

Self−explanatory.

Dump custom fix does not compute per−atom info

Self−explanatory.

Dump custom fix does not compute per−atom vector

Self−explanatory.

Dump custom fix vector is accessed out−of−range

Self−explanatory.

Dump custom variable is not atom−style variable

Only atom−style variables generate per−atom quantities, needed for dump output.

Dump dcd of non−matching # of atoms

Every snapshot written by dump dcd must contain the same # of atoms.

Dump dcd requires sorting by atom ID

Use the dump_modify sort command to enable this.

Dump every variable returned a bad timestep

The variable must return a timestep greater than the current timestep.

Dump local and fix not computed at compatible times

The fix must produce per−atom quantities on timesteps that dump local needs them.

Dump local attributes contain no compute or fix
Self−explanatory.
Dump local cannot sort by atom ID
This is because dump local does not really dump per−atom info.
Dump local compute does not calculate local array
Self−explanatory.
Dump local compute does not calculate local vector
Self−explanatory.
Dump local compute does not compute local info
Self−explanatory.
Dump local compute vector is accessed out−of−range
Self−explanatory.
Dump local count is not consistent across input fields
Every column of output must be the same length.
Dump local fix does not compute local array
Self−explanatory.
Dump local fix does not compute local info
Self−explanatory.
Dump local fix does not compute local vector
Self−explanatory.
Dump local fix vector is accessed out−of−range
Self−explanatory.
Dump modify compute ID does not compute per−atom array
Self−explanatory.
Dump modify compute ID does not compute per−atom info
Self−explanatory.
Dump modify compute ID does not compute per−atom vector
Self−explanatory.
Dump modify compute ID vector is not large enough
Self−explanatory.
Dump modify element names do not match atom types
Number of element names must equal number of atom types.
Dump modify fix ID does not compute per−atom array
Self−explanatory.
Dump modify fix ID does not compute per−atom info
Self−explanatory.
Dump modify fix ID does not compute per−atom vector
Self−explanatory.
Dump modify fix ID vector is not large enough
Self−explanatory.
Dump modify variable is not atom−style variable
Self−explanatory.
Dump sort column is invalid
Self−explanatory.
Dump xtc requires sorting by atom ID
Use the dump_modify sort command to enable this.
Dump_modify region ID does not exist
Self−explanatory.
Dumping an atom property that isn't allocated
The chosen atom style does not define the per−atom quantity being dumped.
Dumping an atom quantity that isn't allocated
Only per−atom quantities that are defined for the atom style being used are allowed.

Duplicate particle in PeriDynamic bond − simulation box is too small
This is likely because your box length is shorter than 2 times the bond length.
Electronic temperature dropped below zero
Something has gone wrong with the fix ttm electron temperature model.
Empty brackets in variable
There is no variable syntax that uses empty brackets. Check the variable doc page.
Energy was not tallied on needed timestep
You are using a thermo keyword that requires potentials to have tallied energy, but they didn't on this timestep. See the variable doc page for ideas on how to make this work.
Expected floating point parameter in input script or data file
The quantity being read is an integer on non−numeric value.
Expected floating point parameter in variable definition
The quantity being read is a non−numeric value.
Expected integer parameter in input script or data file
The quantity being read is a floating point or non−numeric value.
Expected integer parameter in variable definition
The quantity being read is a floating point or non−numeric value.
Failed to allocate %d bytes for array %s
Your LAMMPS simulation has run out of memory. You need to run a smaller simulation or on more processors.
Failed to reallocate %d bytes for array %s
Your LAMMPS simulation has run out of memory. You need to run a smaller simulation or on more processors.
Fewer SRD bins than processors in some dimension
This is not allowed. Make your SRD bin size smaller.
Final box dimension due to fix deform is < 0.0
Self−explanatory.
Fix gpu split must be positive for hybrid pair styles.
See documentation for fix gpu.
Fix ID for compute atom/molecule does not exist
Self−explanatory.
Fix ID for compute reduce does not exist
Self−explanatory.
Fix ID for fix ave/atom does not exist
Self−explanatory.
Fix ID for fix ave/correlate does not exist
Self−explanatory.
Fix ID for fix ave/histo does not exist
Self−explanatory.
Fix ID for fix ave/spatial does not exist
Self−explanatory.
Fix ID for fix ave/time does not exist
Self−explanatory.
Fix ID for fix store/state does not exist
Self−explanatory
Fix ID must be alphanumeric or underscore characters
Self−explanatory.
Fix SRD cannot have both atom attributes angmom and omega
Use either spheroid solute particles or fully generalized aspherical solute particles.
Fix SRD no−slip requires atom attribute torque
This is because the SRD collisions will impart torque to the solute particles.
Fix SRD requires atom attribute angmom or omega

This is because the SRD collisions with cause the solute particles to rotate.

Fix SRD requires atom attribute radius or shape

This is because the solute particles must be finite−size particles, not point particles.

Fix SRD: bad bin assignment for SRD advection

Something has gone wrong in your SRD model; try using more conservative settings.

Fix SRD: bad search bin assignment

Something has gone wrong in your SRD model; try using more conservative settings.

Fix SRD: bad stencil bin for big particle

Something has gone wrong in your SRD model; try using more conservative settings.

Fix SRD: too many big particles in bin

Reset the ATOMPERBIN parameter at the top of fix_srd.cpp to a larger value, and re−compile the code.

Fix SRD: too many walls in bin

This should not happen unless your system has been setup incorrectly.

Fix adapt kspace style does not exist

Self−explanatory.

Fix adapt pair style does not exist

Self−explanatory

Fix adapt pair style param not supported

The pair style does not know about the parameter you specified.

Fix adapt requires atom attribute diameter

The atom style being used does not specify an atom diameter.

Fix adapt type pair range is not valid for pair hybrid sub−style

Self−explanatory.

Fix ave/atom compute array is accessed out−of−range

Self−explanatory.

Fix ave/atom compute does not calculate a per−atom array

Self−explanatory.

Fix ave/atom compute does not calculate a per−atom vector

A compute used by fix ave/atom must generate per−atom values.

Fix ave/atom compute does not calculate per−atom values

A compute used by fix ave/atom must generate per−atom values.

Fix ave/atom fix array is accessed out−of−range

Self−explanatory.

Fix ave/atom fix does not calculate a per−atom array

Self−explanatory.

Fix ave/atom fix does not calculate a per−atom vector

A fix used by fix ave/atom must generate per−atom values.

Fix ave/atom fix does not calculate per−atom values

A fix used by fix ave/atom must generate per−atom values.

Fix ave/atom variable is not atom−style variable

A variable used by fix ave/atom must generate per−atom values.

Fix ave/histo cannot input local values in scalar mode

Self−explanatory.

Fix ave/histo cannot input per−atom values in scalar mode

Self−explanatory.

Fix ave/histo compute array is accessed out−of−range

Self−explanatory.

Fix ave/histo compute does not calculate a global array

Self−explanatory.

Fix ave/histo compute does not calculate a global scalar

Self−explanatory.

Fix ave/histo compute does not calculate a global vector

Self−explanatory.

Fix ave/histo compute does not calculate a local array

Self−explanatory.

Fix ave/histo compute does not calculate a local vector

Self−explanatory.

Fix ave/histo compute does not calculate a per−atom array

Self−explanatory.

Fix ave/histo compute does not calculate a per−atom vector

Self−explanatory.

Fix ave/histo compute does not calculate local values

Self−explanatory.

Fix ave/histo compute does not calculate per−atom values

Self−explanatory.

Fix ave/histo compute vector is accessed out−of−range

Self−explanatory.

Fix ave/histo fix array is accessed out−of−range

Self−explanatory.

Fix ave/histo fix does not calculate a global array

Self−explanatory.

Fix ave/histo fix does not calculate a global scalar

Self−explanatory.

Fix ave/histo fix does not calculate a global vector

Self−explanatory.

Fix ave/histo fix does not calculate a local array

Self−explanatory.

Fix ave/histo fix does not calculate a local vector

Self−explanatory.

Fix ave/histo fix does not calculate a per−atom array

Self−explanatory.

Fix ave/histo fix does not calculate a per−atom vector

Self−explanatory.

Fix ave/histo fix does not calculate local values

Self−explanatory.

Fix ave/histo fix does not calculate per−atom values

Self−explanatory.

Fix ave/histo fix vector is accessed out−of−range

Self−explanatory.

Fix ave/histo input is invalid compute

Self−explanatory.

Fix ave/histo input is invalid fix

Self−explanatory.

Fix ave/histo input is invalid variable

Self−explanatory.

Fix ave/histo inputs are not all global, peratom, or local

All inputs in a single fix ave/histo command must be of the same style.

Fix ave/spatial compute does not calculate a per−atom array

Self−explanatory.

Fix ave/spatial compute does not calculate a per−atom vector

A compute used by fix ave/spatial must generate per−atom values.

Fix ave/spatial compute does not calculate per−atom values

A compute used by fix ave/spatial must generate per−atom values.

Fix ave/spatial compute vector is accessed out−of−range

The index for the vector is out of bounds.

Fix ave/spatial fix does not calculate a per−atom array

Self−explanatory.

Fix ave/spatial fix does not calculate a per−atom vector

A fix used by fix ave/spatial must generate per−atom values.

Fix ave/spatial fix does not calculate per−atom values

A fix used by fix ave/spatial must generate per−atom values.

Fix ave/spatial fix vector is accessed out−of−range

The index for the vector is out of bounds.

Fix ave/spatial for triclinic boxes requires units reduced

Self−explanatory.

Fix ave/spatial settings invalid with changing box

If the ave setting is "running" or "window" and the box size/shape changes during the simulation, then the units setting must be "reduced", else the number of bins may change.

Fix ave/spatial variable is not atom−style variable

A variable used by fix ave/spatial must generate per−atom values.

Fix ave/time cannot set output array intensive/extensive from these inputs

One of more of the vector inputs has individual elements which are flagged as intensive or extensive. Such an input cannot be flagged as all intensive/extensive when turned into an array by fix ave/time.

Fix ave/time cannot use variable with vector mode

Variables produce scalar values.

Fix ave/time columns are inconsistent lengths

Self−explanatory.

Fix ave/time compute array is accessed out−of−range

Self−explanatory.

Fix ave/time compute does not calculate a scalar

Only computes that calculate a scalar or vector quantity (not a per−atom quantity) can be used with fix ave/time.

Fix ave/time compute does not calculate a vector

Only computes that calculate a scalar or vector quantity (not a per−atom quantity) can be used with fix ave/time.

Fix ave/time compute does not calculate an array

Self−explanatory.

Fix ave/time compute vector is accessed out−of−range

The index for the vector is out of bounds.

Fix ave/time fix array is accessed out−of−range

Self−explanatory.

Fix ave/time fix does not calculate a scalar

A fix used by fix ave/time must generate global values.

Fix ave/time fix does not calculate a vector

A fix used by fix ave/time must generate global values.

Fix ave/time fix does not calculate an array

Self−explanatory.

Fix ave/time fix vector is accessed out−of−range

The index for the vector is out of bounds.

Fix ave/time variable is not equal−style variable

A variable used by fix ave/time must generate a global value.

Fix bond/break requires special_bonds = 0,1,1

This is a restriction of the current fix bond/break implementation.

Fix bond/create cutoff is longer than pairwise cutoff

This is not allowed because bond creation is done using the pairwise neighbor list.

Fix bond/create requires special_bonds coul = 0,1,1

Self−explanatory.

Fix bond/create requires special_bonds lj = 0,1,1

Self−explanatory.

Fix bond/swap cannot use dihedral or improper styles

These styles cannot be defined when using this fix.

Fix bond/swap requires pair and bond styles

Self−explanatory.

Fix bond/swap requires special_bonds = 0,1,1

Self−explanatory.

Fix box/relax generated negative box length

The pressure being applied is likely too large. Try applying it incrementally, to build to the high pressure.

Fix command before simulation box is defined

The fix command cannot be used before a read_data, read_restart, or create_box command.

Fix deform is changing yz by too much with changing xy

When both yz and xy are changing, it induces changes in xz if the box must flip from one tilt extreme to another. Thus it is not allowed for yz to grow so much that a flip is induced.

Fix deform tilt factors require triclinic box

Cannot deform the tilt factors of a simulation box unless it is a triclinic (non−orthogonal) box.

Fix deform volume setting is invalid

Cannot use volume style unless other dimensions are being controlled.

Fix deposit region cannot be dynamic

Only static regions can be used with fix deposit.

Fix deposit region does not support a bounding box

Not all regions represent bounded volumes. You cannot use such a region with the fix deposit command.

Fix efield requires atom attribute q

Self−explanatory.

Fix evaporate molecule requires atom attribute molecule

The atom style being used does not define a molecule ID.

Fix external callback function not set

This must be done by an external program in order to use this fix.

Fix for fix ave/atom not computed at compatible time

Fixes generate their values on specific timesteps. Fix ave/atom is requesting a value on a non−allowed timestep.

Fix for fix ave/correlate not computed at compatible time

Fixes generate their values on specific timesteps. Fix ave/correlate is requesting a value on a non−allowed timestep.

Fix for fix ave/histo not computed at compatible time

Fixes generate their values on specific timesteps. Fix ave/histo is requesting a value on a non−allowed timestep.

Fix for fix ave/spatial not computed at compatible time

Fixes generate their values on specific timesteps. Fix ave/spatial is requesting a value on a non−allowed timestep.

Fix for fix ave/time not computed at compatible time

Fixes generate their values on specific timesteps. Fix ave/time is requesting a value on a non−allowed timestep.

Fix for fix store/state not computed at compatible time

Fixes generate their values on specific timesteps. Fix store/state is requesting a value on a non−allowed timestep.

Fix freeze requires atom attribute torque

The atom style defined does not have this attribute.

Fix heat group has no atoms

Self−explanatory.

Fix heat kinetic energy went negative

This will cause the velocity rescaling about to be performed by fix heat to be invalid.

Fix in variable not computed at compatible time

Fixes generate their values on specific timesteps. The variable is requesting the values on a non−allowed timestep.

Fix langevin period must be > 0.0

The time window for temperature relaxation must be > 0

Fix momentum group has no atoms

Self−explanatory.

Fix move cannot define z or vz variable for 2d problem

Self−explanatory.

Fix move cannot have 0 length rotation vector

Self−explanatory.

Fix move cannot rotate aroung non z−axis for 2d problem

Self−explanatory.

Fix move cannot set linear z motion for 2d problem

Self−explanatory.

Fix move cannot set wiggle z motion for 2d problem

Self−explanatory.

Fix msst compute ID does not compute potential energy

Self−explanatory.

Fix msst compute ID does not compute pressure

Self−explanatory.

Fix msst compute ID does not compute temperature

Self−explanatory.

Fix msst requires a periodic box

Self−explanatory.

Fix msst tscale must satisfy 0 <= tscale < 1

Self−explanatory.

Fix npt/nph has tilted box too far − box flips are not yet implemented

This feature has not yet been added. However, if you are applying an off−diagonal pressure to a fluid, the box may want to tilt indefinitely, because the fluid cannot support the pressure you are imposing.

Fix nve/asphere cannot be used with atom attributes diameter or rmass

These attributes override the shape and mass settings, so cannot be used.

Fix nve/asphere requires atom attributes angmom, quat, torque, shape

An atom style that specifies these quantities is needed.

Fix nve/asphere requires extended particles

This fix can only be used for particles with a shape setting.

Fix nve/sphere requires atom attribute diameter or shape

An atom style that specifies these quantities is needed.

Fix nve/sphere requires atom attribute mu

An atom style with this attribute is needed.

Fix nve/sphere requires atom attributes omega, torque

An atom style with these attributes is needed.

Fix nve/sphere requires extended particles

This fix can only be used for particles of a finite size.

Fix nve/sphere requires spherical particle shapes

Self−explanatory.

Fix nvt/nph/npt asphere cannot be used with atom attributes diameter or rmass

Those attributes are for spherical particles.

Fix nvt/nph/npt asphere requires atom attributes quat, angmom, torque, shape

Those attributes are what are used to define aspherical particles.

Fix nvt/nph/npt asphere requires extended particles
The shape setting for a particle in the fix group has shape = 0.0, which means it is a point particle.
Fix nvt/nph/npt sphere requires atom attribute diameter or shape
An atom style that specifies these quantities is needed.
Fix nvt/nph/npt sphere requires atom attributes omega, torque
Those attributes are what are used to define spherical particles.
Fix nvt/npt/nph damping parameters must be > 0.0
Self−explanatory.
Fix nvt/sphere requires extended particles
This fix can only be used for particles of a finite size.
Fix nvt/sphere requires spherical particle shapes
Self−explanatory.
Fix orient/fcc file open failed
The fix orient/fcc command could not open a specified file.
Fix orient/fcc file read failed
The fix orient/fcc command could not read the needed parameters from a specified file.
Fix orient/fcc found self twice
The neighbor lists used by fix orient/fcc are messed up. If this error occurs, it is likely a bug, so send an email to the developers.
Fix peri neigh does not exist
Somehow a fix that the pair style defines has been deleted.
Fix pour region ID does not exist
Self−explanatory.
Fix pour region cannot be dynamic
Only static regions can be used with fix pour.
Fix pour region does not support a bounding box
Not all regions represent bounded volumes. You cannot use such a region with the fix pour command.
Fix pour requires atom attributes radius, rmass
The atom style defined does not have these attributes.
Fix press/berendsen damping parameters must be > 0.0
Self−explanatory.
Fix qeq/comb group has no atoms
Self−explanatory.
Fix qeq/comb requires atom attribute q
An atom style with charge must be used to perform charge equilibration.
Fix reax/bonds numbonds > nsbmax_most
The limit of the number of bonds expected by the ReaxFF force field was exceeded.
Fix recenter group has no atoms
Self−explanatory.
Fix rigid atom has non−zero image flag in a non−periodic dimension
You cannot set image flags for non−periodic dimensions.
Fix rigid molecule requires atom attribute molecule
Self−explanatory.
Fix rigid/nvt period must be > 0.0
Self−explanatory
Fix rigid: Bad principal moments
The principal moments of inertia computed for a rigid body are not within the required tolerances.
Fix shake cannot be used with minimization
Cannot use fix shake while doing an energy minimization since it turns off bonds that should contribute to the energy.
Fix spring couple group ID does not exist
Self−explanatory.

Fix srd lamda must be >= 0.6 of SRD grid size
: This is a requirement for accuracy reasons.

Fix srd requires SRD particles all have same mass
: Self−explanatory.

Fix srd requires ghost atoms store velocity
: Use the communicate vel yes command to enable this.

Fix srd requires newton pair on
: Self−explanatory.

Fix store/state compute array is accessed out−of−range
: Self−explanatory.

Fix store/state compute does not calculate a per−atom array
: The compute calculates a per−atom vector.

Fix store/state compute does not calculate a per−atom vector
: The compute calculates a per−atom vector.

Fix store/state compute does not calculate per−atom values
: Computes that calculate global or local quantities cannot be used with fix store/state.

Fix store/state fix array is accessed out−of−range
: Self−explanatory.

Fix store/state fix does not calculate a per−atom array
: The fix calculates a per−atom vector.

Fix store/state fix does not calculate a per−atom vector
: The fix calculates a per−atom array.

Fix store/state fix does not calculate per−atom values
: Fixes that calculate global or local quantities cannot be used with fix store/state.

Fix store/state for atom property that isn't allocated
: Self−explanatory.

Fix store/state variable is not atom−style variable
: Only atom−style variables calculate per−atom quantities.

Fix temp/berendsen period must be > 0.0
: Self−explanatory.

Fix thermal/conductivity swap value must be positive
: Self−explanatory.

Fix tmd must come after integration fixes
: Any fix tmd command must appear in the input script after all time integration fixes (nve, nvt, npt). See the fix tmd documentation for details.

Fix ttm electron temperatures must be > 0.0
: Self−explanatory.

Fix ttm electronic_density must be > 0.0
: Self−explanatory.

Fix ttm electronic_specific_heat must be > 0.0
: Self−explanatory.

Fix ttm electronic_thermal_conductivity must be >= 0.0
: Self−explanatory.

Fix ttm gamma_p must be > 0.0
: Self−explanatory.

Fix ttm gamma_s must be >= 0.0
: Self−explanatory.

Fix ttm number of nodes must be > 0
: Self−explanatory.

Fix ttm v_0 must be >= 0.0
: Self−explanatory.

Fix used in compute atom/molecule not computed at compatible time

The fix must produce per−atom quantities on timesteps that the compute needs them.

Fix used in compute reduce not computed at compatible time

Fixes generate their values on specific timesteps. Compute sum is requesting a value on a non−allowed timestep.

Fix viscosity swap value must be positive

Self−explanatory.

Fix viscosity vtarget value must be positive

Self−explanatory.

Fix wall cutoff <= 0.0

Self−explanatory.

Fix wall/colloid cannot be used with atom attribute diameter

Only finite−size particles defined by the shape command can be used.

Fix wall/colloid requires atom attribute shape

Self−explanatory.

Fix wall/colloid requires extended particles

Self−explanatory.

Fix wall/colloid requires spherical particles

Self−explanatory.

Fix wall/gran is incompatible with Pair style

Must use a granular pair style to define the parameters needed for this fix.

Fix wall/gran requires atom attributes radius, omega, torque

The atom style defined does not have these attributes.

Fix wall/region colloid cannot be used with atom attribute diameter

Only finite−size particles defined by the shape command can be used.

Fix wall/region colloid requires atom attribute shape

Self−explanatory.

Fix wall/region colloid requires extended particles

Self−explanatory.

Fix wall/region colloid requires spherical particles

Self−explanatory.

Fix wall/region cutoff <= 0.0

Self−explanatory.

Fix_modify order must be 3 or 5

Self−explanatory.

Fix_modify pressure ID does not compute pressure

The compute ID assigned to the fix must compute pressure.

Fix_modify temperature ID does not compute temperature

The compute ID assigned to the fix must compute temperature.

Found no restart file matching pattern

When using a "*" in the restart file name, no matching file was found.

GPU is not the first fix for this run

This is the way the fix must be defined in your input script.

GPU library not compiled for this accelerator

The GPU library was not built for your accelerator. Check the arch flag in lib/gpu.

Gmask function in equal−style variable formula

Gmask is per−atom operation.

Gravity changed since fix pour was created

Gravity must be static and not dynamic for use with fix pour.

Gravity must point in −y to use with fix pour in 2d

Gravity must be pointing "down" in a 2d box.

Gravity must point in −z to use with fix pour in 3d

Gravity must be pointing "down" in a 3d box, i.e. theta = 180.0.

Grmask function in equal−style variable formula
: Grmask is per−atom operation.

Group ID does not exist
: A group ID used in the group command does not exist.

Group ID in variable formula does not exist
: Self−explanatory.

Group command before simulation box is defined
: The group command cannot be used before a read_data, read_restart, or create_box command.

Group region ID does not exist
: A region ID used in the group command does not exist.

Illegal ... command
: Self−explanatory. Check the input script syntax and compare to the documentation for the command. You can use −echo screen as a command−line option when running LAMMPS to see the offending line.

Illegal COMB parameter
: One or more of the coefficients defined in the potential file is invalid.

Illegal Stillinger−Weber parameter
: One or more of the coefficients defined in the potential file is invalid.

Illegal Tersoff parameter
: One or more of the coefficients defined in the potential file is invalid.

Illegal chemical element names
: The name is too long to be a chemical element.

Illegal fix gpu command
: Self−explanatory.

Illegal number of angle table entries
: There must be at least 2 table entries.

Illegal number of bond table entries
: There must be at least 2 table entries.

Illegal number of pair table entries
: There must be at least 2 table entries.

Illegal simulation box
: The lower bound of the simulation box is greater than the upper bound.

Improper atom missing in delete_bonds
: The delete_bonds command cannot find one or more atoms in a particular improper on a particular processor. The pairwise cutoff is too short or the atoms are too far apart to make a valid improper.

Improper atom missing in set command
: The set command cannot find one or more atoms in a particular improper on a particular processor. The pairwise cutoff is too short or the atoms are too far apart to make a valid improper.

Improper atoms %d %d %d %d missing on proc %d at step
: One or more of 4 atoms needed to compute a particular improper are missing on this processor. Typically this is because the pairwise cutoff is set too short or the improper has blown apart and an atom is too far away.

Improper coeff for hybrid has invalid style
: Improper style hybrid uses another improper style as one of its coefficients. The improper style used in the improper_coeff command or read from a restart file is not recognized.

Improper coeffs are not set
: No improper coefficients have been assigned in the data file or via the improper_coeff command.

Improper style hybrid cannot have hybrid as an argument
: Self−explanatory.

Improper style hybrid cannot have none as an argument
: Self−explanatory.

Improper style hybrid cannot use same improper style twice
: Self−explanatory.

Improper_coeff command before improper_style is defined
Coefficients cannot be set in the data file or via the improper_coeff command until an improper_style has been assigned.
Improper_coeff command before simulation box is defined
The improper_coeff command cannot be used before a read_data, read_restart, or create_box command.
Improper_coeff command when no impropers allowed
The chosen atom style does not allow for impropers to be defined.
Improper_style command when no impropers allowed
The chosen atom style does not allow for impropers to be defined.
Impropers assigned incorrectly
Impropers read in from the data file were not assigned correctly to atoms. This means there is something invalid about the topology definitions.
Impropers defined but no improper types
The data file header lists improper but no improper types.
Inconsistent iparam/jparam values in fix bond/create command
If itype and jtype are the same, then their maxbond and newtype settings must also be the same.
Incorrect args for angle coefficients
Self−explanatory. Check the input script or data file.
Incorrect args for bond coefficients
Self−explanatory. Check the input script or data file.
Incorrect args for dihedral coefficients
Self−explanatory. Check the input script or data file.
Incorrect args for improper coefficients
Self−explanatory. Check the input script or data file.
Incorrect args for pair coefficients
Self−explanatory. Check the input script or data file.
Incorrect args in pair_style command
Self−explanatory.
Incorrect atom format in data file
Number of values per atom line in the data file is not consistent with the atom style.
Incorrect boundaries with slab Ewald
Must have periodic x,y dimensions and non−periodic z dimension to use 2d slab option with Ewald.
Incorrect boundaries with slab PPPM
Must have periodic x,y dimensions and non−periodic z dimension to use 2d slab option with PPPM.
Incorrect element names in EAM potential file
The element names in the EAM file do not match those requested.
Incorrect format in COMB potential file
Incorrect number of words per line in the potential file.
Incorrect format in MEAM potential file
Incorrect number of words per line in the potential file.
Incorrect format in NEB coordinate file
Self−explanatory.
Incorrect format in Stillinger−Weber potential file
Incorrect number of words per line in the potential file.
Incorrect format in TMD target file
Format of file read by fix tmd command is incorrect.
Incorrect format in Tersoff potential file
Incorrect number of words per line in the potential file.
Incorrect multiplicity arg for dihedral coefficients
Self−explanatory. Check the input script or data file.
Incorrect sign arg for dihedral coefficients
Self−explanatory. Check the input script or data file.

Incorrect velocity format in data file
Each atom style defines a format for the Velocity section of the data file. The read−in lines do not match.
Incorrect weight arg for dihedral coefficients
Self−explanatory. Check the input script or data file.
Index between variable brackets must be positive
Self−explanatory.
Indexed per−atom vector in variable formula without atom map
Accessing a value from an atom vector requires the ability to lookup an atom index, which is provided by an atom map. An atom map does not exist (by default) for non−molecular problems. Using the atom_modify map command will force an atom map to be created.
Induced tilt by displace_box is too large
The final tilt value must be between −1/2 and 1/2 of the perpendicular box length.
Initial temperatures not all set in fix ttm
Self−explantory.
Input line too long after variable substitution
This is a hard (very large) limit defined in the input.cpp file.
Input line too long: %s
This is a hard (very large) limit defined in the input.cpp file.
Insertion region extends outside simulation box
Region specified with fix pour command extends outside the global simulation box.
Insufficient Jacobi rotations for POEMS body
Eigensolve for rigid body was not sufficiently accurate.
Insufficient Jacobi rotations for rigid body
Eigensolve for rigid body was not sufficiently accurate.
Insufficient memory on accelerator.
Self−explanatory.
Invalid Boolean syntax in if command
Self−explanatory.
Invalid REAX atom type
There is a mis−match between LAMMPS atom types and the elements listed in the ReaxFF force field file.
Invalid angle style
The choice of angle style is unknown.
Invalid angle table length
Length must be 2 or greater.
Invalid angle type in Angles section of data file
Angle type must be positive integer and within range of specified angle types.
Invalid angle type index for fix shake
Self−explanatory.
Invalid atom ID in Angles section of data file
Atom IDs must be positive integers and within range of defined atoms.
Invalid atom ID in Atoms section of data file
Atom IDs must be positive integers.
Invalid atom ID in Bonds section of data file
Atom IDs must be positive integers and within range of defined atoms.
Invalid atom ID in Dihedrals section of data file
Atom IDs must be positive integers and within range of defined atoms.
Invalid atom ID in Impropers section of data file
Atom IDs must be positive integers and within range of defined atoms.
Invalid atom ID in Velocities section of data file
Atom IDs must be positive integers and within range of defined atoms.
Invalid atom mass for fix shake

Mass specified in fix shake command must be > 0.0.

Invalid atom style

The choice of atom style is unknown.

Invalid atom type in Atoms section of data file

Atom types must range from 1 to specified # of types.

Invalid atom type in create_atoms command

The create_box command specified the range of valid atom types. An invalid type is being requested.

Invalid atom type in fix bond/create command

Self−explanatory.

Invalid atom type in neighbor exclusion list

Atom types must range from 1 to Ntypes inclusive.

Invalid atom type index for fix shake

Atom types must range from 1 to Ntypes inclusive.

Invalid atom types in pair_write command

Atom types must range from 1 to Ntypes inclusive.

Invalid atom vector in variable formula

The atom vector is not recognized.

Invalid attribute in dump custom command

Self−explantory.

Invalid attribute in dump local command

Self−explantory.

Invalid attribute in dump modify command

Self−explantory.

Invalid bond style

The choice of bond style is unknown.

Invalid bond table length

Length must be 2 or greater.

Invalid bond type in Bonds section of data file

Bond type must be positive integer and within range of specified bond types.

Invalid bond type in fix bond/break command

Self−explanatory.

Invalid bond type in fix bond/create command

Self−explanatory.

Invalid bond type index for fix shake

Self−explanatory. Check the fix shake command in the input script.

Invalid coeffs for this dihedral style

Cannot set class 2 coeffs in data file for this dihedral style.

Invalid command−line argument

One or more command−line arguments is invalid. Check the syntax of the command you are using to launch LAMMPS.

Invalid compute ID in variable formula

The compute is not recognized.

Invalid compute style

Self−explanatory.

Invalid cutoff in communicate command

Specified cutoff must be >= 0.0.

Invalid cutoffs in pair_write command

Inner cutoff must be larger than 0.0 and less than outer cutoff.

Invalid d1 or d2 value for pair colloid coeff

Neither d1 or d2 can be < 0.

Invalid data file section: Angle Coeffs

Atom style does not allow angles.

Invalid data file section: AngleAngle Coeffs
Atom style does not allow impropers.
Invalid data file section: AngleAngleTorsion Coeffs
Atom style does not allow dihedrals.
Invalid data file section: AngleTorsion Coeffs
Atom style does not allow dihedrals.
Invalid data file section: Angles
Atom style does not allow angles.
Invalid data file section: Bond Coeffs
Atom style does not allow bonds.
Invalid data file section: BondAngle Coeffs
Atom style does not allow angles.
Invalid data file section: BondBond Coeffs
Atom style does not allow angles.
Invalid data file section: BondBond13 Coeffs
Atom style does not allow dihedrals.
Invalid data file section: Bonds
Atom style does not allow bonds.
Invalid data file section: Dihedral Coeffs
Atom style does not allow dihedrals.
Invalid data file section: Dihedrals
Atom style does not allow dihedrals.
Invalid data file section: EndBondTorsion Coeffs
Atom style does not allow dihedrals.
Invalid data file section: Improper Coeffs
Atom style does not allow impropers.
Invalid data file section: Impropers
Atom style does not allow impropers.
Invalid data file section: MiddleBondTorsion Coeffs
Atom style does not allow dihedrals.
Invalid delta_conf in tad command
The value must be between 0 and 1 inclusive.
Invalid density in Atoms section of data file
Density value cannot be <= 0.0.
Invalid dihedral style
The choice of dihedral style is unknown.
Invalid dihedral type in Dihedrals section of data file
Dihedral type must be positive integer and within range of specified dihedral types.
Invalid dipole line in data file
Self−explanatory.
Invalid dipole value
Self−explanatory.
Invalid dump dcd filename
Filenames used with the dump dcd style cannot be binary or compressed or cause multiple files to be written.
Invalid dump frequency
Dump frequency must be 1 or greater.
Invalid dump style
The choice of dump style is unknown.
Invalid dump xtc filename
Filenames used with the dump xtc style cannot be binary or compressed or cause multiple files to be written.

Invalid dump xyz filename
Filenames used with the dump xyz style cannot be binary or cause files to be written by each processor.
Invalid dump_modify threshhold operator
Operator keyword used for threshold specification in not recognized.
Invalid fix ID in variable formula
The fix is not recognized.
Invalid fix ave/time off column
Self−explantory.
Invalid fix box/relax command for a 2d simulation
Fix box/relax styles involving the z dimension cannot be used in a 2d simulation.
Invalid fix box/relax command pressure settings
If multiple dimensions are coupled, those dimensions must be specified.
Invalid fix box/relax pressure settings
Settings for coupled dimensions must be the same.
Invalid fix nvt/npt/nph command for a 2d simulation
Cannot control z dimension in a 2d model.
Invalid fix nvt/npt/nph command pressure settings
If multiple dimensions are coupled, those dimensions must be specified.
Invalid fix nvt/npt/nph pressure settings
Settings for coupled dimensions must be the same.
Invalid fix press/berendsen for a 2d simulation
The z component of pressure cannot be controlled for a 2d model.
Invalid fix press/berendsen pressure settings
Settings for coupled dimensions must be the same.
Invalid fix style
The choice of fix style is unknown.
Invalid flag in force field section of restart file
Unrecognized entry in restart file.
Invalid flag in header section of restart file
Unrecognized entry in restart file.
Invalid flag in type arrays section of restart file
Unrecognized entry in restart file.
Invalid frequency in temper command
Nevery must be > 0.
Invalid group ID in neigh_modify command
A group ID used in the neigh_modify command does not exist.
Invalid group function in variable formula
Group function is not recognized.
Invalid group in communicate command
Self−explanatory.
Invalid improper style
The choice of improper style is unknown.
Invalid improper type in Impropers section of data file
Improper type must be positive integer and within range of specified improper types.
Invalid keyword in angle table parameters
Self−explanatory.
Invalid keyword in bond table parameters
Self−explanatory.
Invalid keyword in compute angle/local command
Self−explanatory.
Invalid keyword in compute bond/local command
Self−explanatory.

Invalid keyword in compute dihedral/local command
Self−explanatory.
Invalid keyword in compute improper/local command
Self−explanatory.
Invalid keyword in compute pair/local command
Self−explanatory.
Invalid keyword in compute property/atom command
Self−explanatory.
Invalid keyword in compute property/local command
Self−explanatory.
Invalid keyword in compute property/molecule command
Self−explanatory.
Invalid keyword in dump cfg command
Self−explanatory.
Invalid keyword in pair table parameters
Keyword used in list of table parameters is not recognized.
Invalid keyword in thermo_style custom command
One or more specified keywords are not recognized.
Invalid kspace style
The choice of kspace style is unknown.
Invalid mass line in data file
Self−explanatory.
Invalid mass value
Self−explanatory.
Invalid math function in variable formula
Self−explanatory.
Invalid math/group/special function in variable formula
Self−explanatory.
Invalid option in lattice command for non−custom style
Certain lattice keywords are not supported unless the lattice style is "custom".
Invalid order of forces within respa levels
For respa, ordering of force computations within respa levels must obey certain rules. E.g. bonds cannot be compute less frequently than angles, pairwise forces cannot be computed less frequently than kspace, etc.
Invalid pair style
The choice of pair style is unknown.
Invalid pair table cutoff
Cutoffs in pair_coeff command are not valid with read−in pair table.
Invalid pair table length
Length of read−in pair table is invalid
Invalid radius in Atoms section of data file
Radius must be >= 0.0.
Invalid random number seed in fix ttm command
Random number seed must be > 0.
Invalid random number seed in set command
Random number seed must be > 0.
Invalid region style
The choice of region style is unknown.
Invalid replace values in compute reduce
Self−explanatory.
Invalid run command N value
The number of timesteps must fit in a 32−bit integer. If you want to run for more steps than this, perform

multiple shorter runs.

Invalid run command start/stop value
Self−explanatory.

Invalid run command upto value
Self−explanatory.

Invalid seed for Marsaglia random # generator
The initial seed for this random number generator must be a positive integer less than or equal to 900 million.

Invalid seed for Park random # generator
The initial seed for this random number generator must be a positive integer.

Invalid shape line in data file
Self−explanatory.

Invalid shape line in data file
Self−explanatory.

Invalid shape value
Self−explanatory.

Invalid shear direction for fix wall/gran
Self−explanatory.

Invalid special function in variable formula
Self−explanatory.

Invalid style in pair_write command
Self−explanatory. Check the input script.

Invalid syntax in variable formula
Self−explanatory.

Invalid t_event in prd command
Self−explanatory.

Invalid t_event in tad command
The value must be greater than 0.

Invalid thermo keyword in variable formula
The keyword is not recognized.

Invalid tmax in tad command
The value must be greater than 0.0.

Invalid type for dipole set
Dipole command must set a type from 1−N where N is the number of atom types.

Invalid type for mass set
Mass command must set a type from 1−N where N is the number of atom types.

Invalid type for shape set
Atom type is out of bounds.

Invalid value in set command
The value specified for the setting is invalid, likely because it is too small or too large.

Invalid variable evaluation in variable formula
A variable used in a formula could not be evaluated.

Invalid variable in next command
Self−explanatory.

Invalid variable name in variable formula
Variable name is not recognized.

Invalid variable name
Variable name used in an input script line is invalid.

Invalid variable style with next command
Variable styles *equal* and *world* cannot be used in a next command.

Invalid wiggle direction for fix wall/gran
Self−explanatory.

Invoked angle equil angle on angle style none
Self−explanatory.
Invoked angle single on angle style none
Self−explanatory.
Invoked bond equil distance on bond style none
Self−explanatory.
Invoked bond single on bond style none
Self−explanatory.
Invoked pair single on pair style none
A command (e.g. a dump) attempted to invoke the single() function on a pair style none, which is illegal. You are probably attempting to compute per−atom quantities with an undefined pair style.
KSpace style has not yet been set
Cannot use kspace_modify command until a kspace style is set.
KSpace style is incompatible with Pair style
Setting a kspace style requires that a pair style with a long−range Coulombic component be selected.
Keyword %s in MEAM parameter file not recognized
Self−explanatory.
Kspace style requires atom attribute q
The atom style defined does not have these attributes.
Label wasn't found in input script
Self−explanatory.
Lattice orient vectors are not orthogonal
The three specified lattice orientation vectors must be mutually orthogonal.
Lattice orient vectors are not right−handed
The three specified lattice orientation vectors must create a right−handed coordinate system such that a1 cross a2 = a3.
Lattice primitive vectors are collinear
The specified lattice primitive vectors do not for a unit cell with non−zero volume.
Lattice settings are not compatible with 2d simulation
One or more of the specified lattice vectors has a non−zero z component.
Lattice spacings are invalid
Each x,y,z spacing must be > 0.
Lattice style incompatible with simulation dimension
2d simulation can use sq, sq2, or hex lattice. 3d simulation can use sc, bcc, or fcc lattice.
Log of zero/negative value in variable formula
Self−explanatory.
MEAM library error %d
A call to the MEAM Fortran library returned an error.
MPI_LMP_BIGINT and bigint in lmptype.h are not compatible
The size of the MPI datatype does not match the size of a bigint.
MPI_LMP_TAGINT and tagint in lmptype.h are not compatible
The size of the MPI datatype does not match the size of a tagint.
Mass command before simulation box is defined
The mass command cannot be used before a read_data, read_restart, or create_box command.
Min_style command before simulation box is defined
The min_style command cannot be used before a read_data, read_restart, or create_box command.
Minimization could not find thermo_pe compute
This compute is created by the thermo command. It must have been explicitly deleted by a uncompute command.
Minimize command before simulation box is defined
The minimize command cannot be used before a read_data, read_restart, or create_box command.
Mismatched brackets in variable

Self−explanatory.

Mismatched compute in variable formula

A compute is referenced incorrectly or a compute that produces per−atom values is used in an equal−style variable formula.

Mismatched fix in variable formula

A fix is referenced incorrectly or a fix that produces per−atom values is used in an equal−style variable formula.

Mismatched variable in variable formula

A variable is referenced incorrectly or an atom−style variable that produces per−atom values is used in an equal−style variable formula.

Molecular data file has too many atoms

These kids of data files are currently limited to a number of atoms that fits in a 32−bit integer.

Molecule count changed in compute atom/molecule

Number of molecules must remain constant over time.

Molecule count changed in compute com/molecule

Number of molecules must remain constant over time.

Molecule count changed in compute gyration/molecule

Number of molecules must remain constant over time.

Molecule count changed in compute msd/molecule

Number of molecules must remain constant over time.

Molecule count changed in compute property/molecule

Number of molecules must remain constant over time.

More than one fix deform

Only one fix deform can be defined at a time.

More than one fix freeze

Only one of these fixes can be defined, since the granular pair potentials access it.

More than one fix shake

Only one fix shake can be defined.

Must define angle_style before Angle Coeffs

Must use an angle_style command before reading a data file that defines Angle Coeffs.

Must define angle_style before BondAngle Coeffs

Must use an angle_style command before reading a data file that defines Angle Coeffs.

Must define angle_style before BondBond Coeffs

Must use an angle_style command before reading a data file that defines Angle Coeffs.

Must define bond_style before Bond Coeffs

Must use a bond_style command before reading a data file that defines Bond Coeffs.

Must define dihedral_style before AngleAngleTorsion Coeffs

Must use a dihedral_style command before reading a data file that defines AngleAngleTorsion Coeffs.

Must define dihedral_style before AngleTorsion Coeffs

Must use a dihedral_style command before reading a data file that defines AngleTorsion Coeffs.

Must define dihedral_style before BondBond13 Coeffs

Must use a dihedral_style command before reading a data file that defines BondBond13 Coeffs.

Must define dihedral_style before Dihedral Coeffs

Must use a dihedral_style command before reading a data file that defines Dihedral Coeffs.

Must define dihedral_style before EndBondTorsion Coeffs

Must use a dihedral_style command before reading a data file that defines EndBondTorsion Coeffs.

Must define dihedral_style before MiddleBondTorsion Coeffs

Must use a dihedral_style command before reading a data file that defines MiddleBondTorsion Coeffs.

Must define improper_style before AngleAngle Coeffs

Must use an improper_style command before reading a data file that defines AngleAngle Coeffs.

Must define improper_style before Improper Coeffs

Must use an improper_style command before reading a data file that defines Improper Coeffs.

Must define pair_style before Pair Coeffs

Must use a pair_style command before reading a data file that defines Pair Coeffs.

Must have more than one processor partition to temper

Cannot use the temper command with only one processor partition. Use the −partition command−line option.

Must read Atoms before Angles

The Atoms section of a data file must come before an Angles section.

Must read Atoms before Bonds

The Atoms section of a data file must come before a Bonds section.

Must read Atoms before Dihedrals

The Atoms section of a data file must come before a Dihedrals section.

Must read Atoms before Impropers

The Atoms section of a data file must come before an Impropers section.

Must read Atoms before Velocities

The Atoms section of a data file must come before a Velocities section.

Must set both respa inner and outer

Cannot use just the inner or outer option with respa without using the other.

Must specify a region in fix deposit

The region keyword must be specified with this fix.

Must specify a region in fix pour

The region keyword must be specified with this fix.

Must use −in switch with multiple partitions

A multi−partition simulation cannot read the input script from stdin. The −in command−line option must be used to specify a file.

Must use a block or cylinder region with fix pour

Self−explanatory.

Must use a block region with fix pour for 2d simulations

Self−explanatory.

Must use a bond style with TIP4P potential

TIP4P potentials assume bond lengths in water are constrained by a fix shake command.

Must use a molecular atom style with fix poems molecule

Self−explanatory.

Must use a z−axis cylinder with fix pour

The axis of the cylinder region used with the fix pour command must be oriented along the z dimension.

Must use an angle style with TIP4P potential

TIP4P potentials assume angles in water are constrained by a fix shake command.

Must use atom style with molecule IDs with fix bond/swap

Self−explanatory.

Must use pair_style comb with fix qeq/comb

Self−explanatory.

Must use variable energy with fix addforce

Must define an energy vartiable when applyting a dynamic force during minimization.

NEB command before simulation box is defined

Self−explanatory.

NEB requires damped dynamics minimizer

Use a different minimization style.

NEB requires use of fix neb

Self−explanatory.

Needed topology not in data file

The header of the data file indicated that bonds or angles or dihedrals or impropers would be included, but they were not present.

Neigh_modify exclude molecule requires atom attribute molecule

Self−explanatory.

Neigh_modify include group != atom_modify first group

Self−explanatory.

Neighbor delay must be 0 or multiple of every setting

The delay and every parameters set via the neigh_modify command are inconsistent. If the delay setting is non−zero, then it must be a multiple of the every setting.

Neighbor include group not allowed with ghost neighbors

This is a current restriction within LAMMPS.

Neighbor list overflow, boost neigh_modify one or page

There are too many neighbors of a single atom. Use the neigh_modify command to increase the neighbor page size and the max number of neighbors allowed for one atom.

Neighbor multi not yet enabled for ghost neighbors

This is a current restriction within LAMMPS.

Neighbor multi not yet enabled for granular

Self−explanatory.

Neighbor multi not yet enabled for rRESPA

Self−explanatory.

Neighbor page size must be >= 10x the one atom setting

This is required to prevent wasting too much memory.

Neighbors of ghost atoms only allowed for full neighbor lists

This is a current restriction within LAMMPS.

New bond exceeded bonds per atom in fix bond/create

See the read_data command for info on setting the "extra bond per atom" header value to allow for additional bonds to be formed.

New bond exceeded special list size in fix bond/create

See the special_bonds extra command for info on how to leave space in the special bonds list to allow for additional bonds to be formed.

Newton bond change after simulation box is defined

The newton command cannot be used to change the newton bond value after a read_data, read_restart, or create_box command.

No angle style is defined for compute angle/local

Self−explanatory.

No angles allowed with this atom style

Self−explanatory. Check data file.

No atoms in data file

The header of the data file indicated that atoms would be included, but they were not present.

No basis atoms in lattice

Basis atoms must be defined for lattice style user.

No bond style is defined for compute bond/local

Self−explanatory.

No bonds allowed with this atom style

Self−explanatory. Check data file.

No dihedral style is defined for compute dihedral/local

Self−explanatory.

No dihedrals allowed with this atom style

Self−explanatory. Check data file.

No dump custom arguments specified

The dump custom command requires that atom quantities be specified to output to dump file.

No dump local arguments specified

Self−explanatory.

No fix gravity defined for fix pour

Cannot add poured particles without gravity to move them.

No improper style is defined for compute improper/local
Self−explanatory.
No impropers allowed with this atom style
Self−explanatory. Check data file.
No matching element in EAM potential file
The EAM potential file does not contain elements that match the requested elements.
No pair hbond/dreiding coefficients set
Self−explanatory.
No pair style defined for compute group/group
Cannot calculate group interactions without a pair style defined.
No pair style is defined for compute pair/local
Self−explanatory.
No pair style is defined for compute property/local
Self−explanatory.
No rigid bodies defined
The fix specification did not end up defining any rigid bodies.
Non digit character between brackets in variable
Self−explantory.
Non integer # of swaps in temper command
Swap frequency in temper command must evenly divide the total # of timesteps.
One or more atoms belong to multiple rigid bodies
Two or more rigid bodies defined by the fix rigid command cannot contain the same atom.
One or zero atoms in rigid body
Any rigid body defined by the fix rigid command must contain 2 or more atoms.
Out of range atoms − cannot compute PPPM
One or more atoms are attempting to map their charge to a PPPM grid point that is not owned by a processor. This is likely for one of two reasons, both of them bad. First, it may mean that an atom near the boundary of a processor's sub−domain has moved more than 1/2 the neighbor skin distance without neighbor lists being rebuilt and atoms being migrated to new processors. This also means you may be missing pairwise interactions that need to be computed. The solution is to change the re−neighboring criteria via the neigh_modify command. The safest settings are "delay 0 every 1 check yes". Second, it may mean that an atom has moved far outside a processor's sub−domain or even the entire simulation box. This indicates bad physics, e.g. due to highly overlapping atoms, too large a timestep, etc.
Overlapping large/large in pair colloid
This potential is infinite when there is an overlap.
Overlapping small/large in pair colloid
This potential is inifinte when there is an overlap.
POEMS fix must come before NPT/NPH fix
NPT/NPH fix must be defined in input script after all poems fixes, else the fix contribution to the pressure virial is incorrect.
PPPM grid is too large
The global PPPM grid is larger than OFFSET in one or more dimensions. OFFSET is currently set to 4096. You likely need to decrease the requested precision.
PPPM order cannot be greater than %d
Self−explanatory.
PPPM order has been reduced to 0
LAMMPS has attempted to reduce the PPPM order to enable the simulation to run, but can reduce the order no further. Try increasing the accuracy of PPPM by reducing the tolerance size, thus inducing a larger PPPM grid.
PRD command before simulation box is defined
The prd command cannot be used before a read_data, read_restart, or create_box command.
PRD nsteps must be multiple of t_event

Self−explanatory.

PRD t_corr must be multiple of t_event

Self−explanatory.

Pair coeff for hybrid has invalid style

Style in pair coeff must have been listed in pair_style command.

Pair cutoff < Respa interior cutoff

One or more pairwise cutoffs are too short to use with the specified rRESPA cutoffs.

Pair dipole/cut requires atom attributes q, mu, torque, dipole

An atom style that specifies these quantities is needed.

Pair distance < table inner cutoff

Two atoms are closer together than the pairwise table allows.

Pair distance > table outer cutoff

Two atoms are further apart than the pairwise table allows.

Pair dpd requires ghost atoms store velocity

Use the communicate vel yes command to enable this.

Pair gayberne cannot be used with atom attribute diameter

Finite−size particles must be defined with the shape command.

Pair gayberne epsilon a,b,c coeffs are not all set

Each atom type involved in pair_style gayberne must have these 3 coefficients set at least once.

Pair gayberne requires atom attributes quat, torque, shape

An atom style that defines these attributes must be used.

Pair granular requires atom attributes radius, omega, torque

The atom style defined does not have these attributes.

Pair granular requires ghost atoms store velocity

Use the communicate vel yes command to enable this.

Pair granular with shear history requires newton pair off

This is a current restriction of the implementation of pair granular styles with history.

Pair hybrid sub−style does not support single call

You are attempting to invoke a single() call on a pair style that doesn't support it.

Pair hybrid sub−style is not used

No pair_coeff command used a sub−style specified in the pair_style command.

Pair inner cutoff < Respa interior cutoff

One or more pairwise cutoffs are too short to use with the specified rRESPA cutoffs.

Pair inner cutoff >= Pair outer cutoff

The specified cutoffs for the pair style are inconsistent.

Pair lubricate cannot be used with atom attributes diameter or rmass

These attributes override the shape and mass settings, so cannot be used.

Pair lubricate requires atom attribute omega or angmom

An atom style that defines these attributes must be used.

Pair lubricate requires atom attributes torque and shape

An atom style that defines these attributes must be used.

Pair lubricate requires extended particles

This pair style can only be used for particles with a shape setting.

Pair lubricate requires ghost atoms store velocity

Use the communicate vel yes command to enable this.

Pair lubricate requires spherical, mono−disperse particles

This is a current restriction of this pair style.

Pair peri lattice is not identical in x, y, and z

The lattice defined by the lattice command must be cubic.

Pair peri requires a lattice be defined

Use the lattice command for this purpose.

Pair peri requires an atom map, see atom_modify

Even for atomic systems, an atom map is required to find Peridynamic bonds. Use the atom_modify command to define one.

Pair resquared cannot be used with atom attribute diameter
This attribute overrides the shape settings, so cannot be used.

Pair resquared epsilon a,b,c coeffs are not all set
Self−explanatory.

Pair resquared epsilon and sigma coeffs are not all set
Self−explanatory.

Pair resquared requires atom attributes quat, torque, shape
An atom style that defines these attributes must be used.

Pair style AIREBO requires atom IDs
This is a requirement to use the AIREBO potential.

Pair style AIREBO requires newton pair on
See the newton command. This is a restriction to use the AIREBO potential.

Pair style COMB requires atom IDs
This is a requirement to use the AIREBO potential.

Pair style COMB requires atom attribute q
Self−explanatory.

Pair style COMB requires newton pair on
See the newton command. This is a restriction to use the COMB potential.

Pair style MEAM requires newton pair on
See the newton command. This is a restriction to use the MEAM potential.

Pair style Stillinger−Weber requires atom IDs
This is a requirement to use the SW potential.

Pair style Stillinger−Weber requires newton pair on
See the newton command. This is a restriction to use the SW potential.

Pair style Tersoff requires atom IDs
This is a requirement to use the Tersoff potential.

Pair style Tersoff requires newton pair on
See the newton command. This is a restriction to use the Tersoff potential.

Pair style born/coul/long requires atom attribute q
An atom style that defines this attribute must be used.

Pair style buck/coul/cut requires atom attribute q
The atom style defined does not have this attribute.

Pair style buck/coul/long requires atom attribute q
The atom style defined does not have these attributes.

Pair style coul/cut requires atom attribute q
The atom style defined does not have these attributes.

Pair style does not support bond_style quartic
The pair style does not have a single() function, so it can not be invoked by bond_style quartic.

Pair style does not support compute group/group
The pair_style does not have a single() function, so it cannot be invokded by the compute group/group command.

Pair style does not support compute pair/local
The pair style does not have a single() function, so it can not be invoked by fix bond/swap.

Pair style does not support compute property/local
The pair style does not have a single() function, so it can not be invoked by fix bond/swap.

Pair style does not support fix bond/swap
The pair style does not have a single() function, so it can not be invoked by fix bond/swap.

Pair style does not support pair_write
The pair style does not have a single() function, so it can not be invoked by pair write.

Pair style does not support rRESPA inner/middle/outer

You are attempting to use rRESPA options with a pair style that does not support them.

Pair style granular with history requires atoms have IDs

Atoms in the simulation do not have IDs, so history effects cannot be tracked by the granular pair potential.

Pair style hbond/dreiding requires an atom map, see atom_modify

Self−explanatory.

Pair style hbond/dreiding requires atom IDs

Self−explanatory.

Pair style hbond/dreiding requires molecular system

Self−explanatory.

Pair style hbond/dreiding requires newton pair on

See the newton command for details.

Pair style hybrid cannot have hybrid as an argument

Self−explanatory.

Pair style hybrid cannot have none as an argument

Self−explanatory.

Pair style hybrid cannot use same pair style twice

The sub−style arguments of pair_style hybrid cannot be duplicated. Check the input script.

Pair style is incompatible with KSpace style

If a pair style with a long−range Coulombic component is selected, then a kspace style must also be used.

Pair style lj/charmm/coul/charmm requires atom attribute q

The atom style defined does not have these attributes.

Pair style lj/charmm/coul/long requires atom attribute q

The atom style defined does not have these attributes.

Pair style lj/class2/coul/cut requires atom attribute q

The atom style defined does not have this attribute.

Pair style lj/class2/coul/long requires atom attribute q

The atom style defined does not have this attribute.

Pair style lj/cut/coul/cut requires atom attribute q

The atom style defined does not have this attribute.

Pair style lj/cut/coul/long requires atom attribute q

The atom style defined does not have this attribute.

Pair style lj/cut/coul/long/tip4p requires atom IDs

There are no atom IDs defined in the system and the TIP4P potential requires them to find O,H atoms with a water molecule.

Pair style lj/cut/coul/long/tip4p requires atom attribute q

The atom style defined does not have these attributes.

Pair style lj/cut/coul/long/tip4p requires newton pair on

This is because the computation of constraint forces within a water molecule adds forces to atoms owned by other processors.

Pair style lj/gromacs/coul/gromacs requires atom attribute q

An atom_style with this attribute is needed.

Pair style peri_lps requires atom style peri

This is because atom style peri stores quantities needed by the peridynamic potential.

Pair style peri_pmb requires atom style peri

This is because atom style peri stores quantities needed by the peridynamic potential.

Pair style reax requires atom IDs

This is a requirement to use the ReaxFF potential.

Pair style reax requires newton pair on

This is a requirement to use the ReaxFF potential.

Pair table cutoffs must all be equal to use with KSpace

When using pair style table with a long−range KSpace solver, the cutoffs for all atom type pairs must all

be the same, since the long−range solver starts at that cutoff.

Pair table parameters did not set N

List of pair table parameters must include N setting.

Pair tersoff/zbl requires metal or real units

This is a current restriction of this pair potential.

Pair yukawa/colloid cannot be used with atom attribute diameter

Only finite−size particles defined by the shape command can be used.

Pair yukawa/colloid requires atom attribute shape

Self−explanatory.

Pair yukawa/colloid requires spherical particles

Self−explanatory.

Pair_coeff command before pair_style is defined

Self−explanatory.

Pair_coeff command before simulation box is defined

The pair_coeff command cannot be used before a read_data, read_restart, or create_box command.

Pair_modify command before pair_style is defined

Self−explanatory.

Pair_write command before pair_style is defined

Self−explanatory.

Particle on or inside fix wall surface

Particles must be "exterior" to the wall in order for energy/force to be calculated.

Particle on or inside surface of region used in fix wall/region

Particles must be "exterior" to the region surface in order for energy/force to be calculated.

Per−atom compute in equal−style variable formula

Equal−style variables cannot use per−atom quantities.

Per−atom energy was not tallied on needed timestep

You are using a thermo keyword that requires potentials to have tallied energy, but they didn't on this timestep. See the variable doc page for ideas on how to make this work.

Per−atom fix in equal−style variable formula

Equal−style variables cannot use per−atom quantities.

Per−atom virial was not tallied on needed timestep

You are using a thermo keyword that requires potentials to have tallied the virial, but they didn't on this timestep. See the variable doc page for ideas on how to make this work.

Per−processor system is too big

The number of owned atoms plus ghost atoms on a single processor must fit in 32−bit integer.

Potential energy ID for fix neb does not exist

Self−explanatory.

Potential file has duplicate entry

The potential file for a SW or Tersoff potential has more than one entry for the same 3 ordered elements.

Potential file is missing an entry

The potential file for a SW or Tersoff potential does not have a needed entry.

Power by 0 in variable formula

Self−explanatory.

Pressure ID for fix box/relax does not exist

The compute ID needed to compute pressure for the fix does not exist.

Pressure ID for fix modify does not exist

Self−explanatory.

Pressure ID for fix npt/nph does not exist

Self−explanatory.

Pressure ID for fix press/berendsen does not exist

The compute ID needed to compute pressure for the fix does not exist.

Pressure ID for thermo does not exist

The compute ID needed to compute pressure for thermodynamics does not exist.

Pressure control can not be used with fix nvt/asphere

Self−explanatory.

Pressure control can not be used with fix nvt/sllod

Self−explanatory.

Pressure control can not be used with fix nvt/sphere

Self−explanatory.

Pressure control can not be used with fix nvt

Self−explanatory.

Pressure control must be used with fix nph/asphere

Self−explanatory.

Pressure control must be used with fix nph/sphere

Self−explanatory.

Pressure control must be used with fix nph

Self−explanatory.

Pressure control must be used with fix npt/asphere

Self−explanatory.

Pressure control must be used with fix npt/sphere

Self−explanatory.

Pressure control must be used with fix npt

Self−explanatory.

Processor count in z must be 1 for 2d simulation

Self−explanatory.

Processor partitions are inconsistent

The total number of processors in all partitions must match the number of processors LAMMPS is running on.

Processors command after simulation box is defined

The processors command cannot be used after a read_data, read_restart, or create_box command.

Quaternion creation numeric error

A numeric error occurred in the creation of a rigid body by the fix rigid command.

R0 < 0 for fix spring command

Equilibrium spring length is invalid.

Reax_defs.h setting for NATDEF is too small

Edit the setting in the ReaxFF library and re−compile the library and re−build LAMMPS.

Reax_defs.h setting for NNEIGHMAXDEF is too small

Edit the setting in the ReaxFF library and re−compile the library and re−build LAMMPS.

Region ID for compute reduce/region does not exist

Self−explanatory.

Region ID for compute temp/region does not exist

Self−explanatory.

Region ID for dump cfg does not exist

Self−explanatory.

Region ID for dump custom does not exist

Self−explanatory.

Region ID for fix addforce does not exist

Self−explanatory.

Region ID for fix ave/spatial does not exist

Self−explanatory.

Region ID for fix aveforce does not exist

Self−explanatory.

Region ID for fix deposit does not exist

Self−explanatory.

Region ID for fix evaporate does not exist
Self−explanatory.
Region ID for fix heat does not exist
Self−explanatory.
Region ID for fix setforce does not exist
Self−explanatory.
Region ID for fix wall/region does not exist
Self−explanatory.
Region ID in variable formula does not exist
Self−explanatory.
Region cannot have 0 length rotation vector
Self−explanatory.
Region intersect region ID does not exist
Self−explanatory.
Region union or intersect cannot be dynamic
The sub−regions can be dynamic, but not the combined region.
Region union region ID does not exist
One or more of the region IDs specified by the region union command does not exist.
Replacing a fix, but new style != old style
A fix ID can be used a 2nd time, but only if the style matches the previous fix. In this case it is assumed you with to reset a fix's parameters. This error may mean you are mistakenly re−using a fix ID when you do not intend to.
Replicate command before simulation box is defined
The replicate command cannot be used before a read_data, read_restart, or create_box command.
Replicate did not assign all atoms correctly
Atoms replicated by the replicate command were not assigned correctly to processors. This is likely due to some atom coordinates being outside a non−periodic simulation box.
Replicated molecular system atom IDs are too big
See the setting for the allowed atom ID size in the src/lmptype.h file.
Replicated system is too big
See the setting for bigint in the src/lmptype.h file.
Resetting timestep is not allowed with fix move
This is because fix move is moving atoms based on elapsed time.
Respa inner cutoffs are invalid
The first cutoff must be <= the second cutoff.
Respa levels must be >= 1
Self−explanatory.
Respa middle cutoffs are invalid
The first cutoff must be <= the second cutoff.
Reuse of compute ID
A compute ID cannot be used twice.
Reuse of dump ID
A dump ID cannot be used twice.
Reuse of region ID
A region ID cannot be used twice.
Rigid body has degenerate moment of inertia
Fix poems will only work with bodies (collections of atoms) that have non−zero principal moments of inertia. This means they must be 3 or more non−collinear atoms, even with joint atoms removed.
Rigid fix must come before NPT/NPH fix
NPT/NPH fix must be defined in input script after all rigid fixes, else the rigid fix contribution to the pressure virial is incorrect.
Rmask function in equal−style variable formula

Rmask is per−atom operation.

Run command before simulation box is defined

The run command cannot be used before a read_data, read_restart, or create_box command.

Run command start value is after start of run

Self−explanatory.

Run command stop value is before end of run

Self−explanatory.

Run_style command before simulation box is defined

The run_style command cannot be used before a read_data, read_restart, or create_box command.

SRD bin size for fix srd differs from user request

Fix SRD had to adjust the bin size to fit the simulation box.

SRD bins for fix srd are not cubic enough

The bin shape is not within tolerance of cubic.

Same dimension twice in fix ave/spatial

Self−explanatory.

Set command before simulation box is defined

The set command cannot be used before a read_data, read_restart, or create_box command.

Set command with no atoms existing

No atoms are yet defined so the set command cannot be used.

Set region ID does not exist

Region ID specified in set command does not exist.

Shake angles have different bond types

All 3−atom angle−constrained SHAKE clusters specified by the fix shake command that are the same angle type, must also have the same bond types for the 2 bonds in the angle.

Shake atoms %d %d %d %d missing on proc %d at step

The 4 atoms in a single shake cluster specified by the fix shake command are not all accessible to a processor. This probably means an atom has moved too far.

Shake atoms %d %d %d missing on proc %d at step

The 3 atoms in a single shake cluster specified by the fix shake command are not all accessible to a processor. This probably means an atom has moved too far.

Shake atoms %d %d missing on proc %d at step

The 2 atoms in a single shake cluster specified by the fix shake command are not all accessible to a processor. This probably means an atom has moved too far.

Shake cluster of more than 4 atoms

A single cluster specified by the fix shake command can have no more than 4 atoms.

Shake clusters are connected

A single cluster specified by the fix shake command must have a single central atom with up to 3 other atoms bonded to it.

Shake determinant = 0.0

The determinant of the matrix being solved for a single cluster specified by the fix shake command is numerically invalid.

Shake fix must come before NPT/NPH fix

NPT fix must be defined in input script after SHAKE fix, else the SHAKE fix contribution to the pressure virial is incorrect.

Shape command before simulation box is defined

Self−explanatory.

Smallint setting in lmptype.h is invalid

It has to be the size of an integer.

Smallint setting in lmptype.h is not compatible

Smallint stored in restart file is not consistent with LAMMPS version you are running.

Sqrt of negative value in variable formula

Self−explanatory.

Substitution for illegal variable

Input script line contained a variable that could not be substituted for.

System in data file is too big

See the setting for bigint in the src/lmptype.h file.

TAD nsteps must be multiple of t_event

Self−explanatory.

TIP4P hydrogen has incorrect atom type

The TIP4P pairwise computation found an H atom whose type does not agree with the specified H type.

TIP4P hydrogen is missing

The TIP4P pairwise computation failed to find the correct H atom within a water molecule.

TMD target file did not list all group atoms

The target file for the fix tmd command did not list all atoms in the fix group.

Tad command before simulation box is defined

Self−explanatory.

Tagint setting in lmptype.h is invalid

Tagint must be as large or larger than smallint.

Tagint setting in lmptype.h is not compatible

Smallint stored in restart file is not consistent with LAMMPS version you are running.

Target temperature for fix nvt/npt/nph cannot be 0.0

Self−explanatory.

Target temperature for fix rigid/nvt cannot be 0.0

Self−explanatory.

Temper command before simulation box is defined

The temper command cannot be used before a read_data, read_restart, or create_box command.

Temperature ID for fix bond/swap does not exist

Self−explanatory.

Temperature ID for fix box/relax does not exist

Self−explanatory.

Temperature ID for fix nvt/nph/npt does not exist

Self−explanatory.

Temperature ID for fix press/berendsen does not exist

Self−explanatory.

Temperature ID for fix temp/berendsen does not exist

Self−explanatory.

Temperature ID for fix temp/rescale does not exist

Self−explanatory.

Temperature control can not be used with fix nph/asphere

Self−explanatory.

Temperature control can not be used with fix nph/sphere

Self−explanatory.

Temperature control can not be used with fix nph

Self−explanatory.

Temperature control must be used with fix npt/asphere

Self−explanatory.

Temperature control must be used with fix npt/sphere

Self−explanatory.

Temperature control must be used with fix npt

Self−explanatory.

Temperature control must be used with fix nvt/asphere

Self−explanatory.

Temperature control must be used with fix nvt/sllod

Self−explanatory.

Temperature control must be used with fix nvt/sphere
Self−explanatory.
Temperature control must be used with fix nvt
Self−explanatory.
Temperature for fix nvt/sllod does not have a bias
The specified compute must compute temperature with a bias.
Tempering could not find thermo_pe compute
This compute is created by the thermo command. It must have been explicitly deleted by a uncompute command.
Tempering fix ID is not defined
The fix ID specified by the temper command does not exist.
Tempering temperature fix is not valid
The fix specified by the temper command is not one that controls temperature (nvt or langevin).
Thermo and fix not computed at compatible times
Fixes generate values on specific timesteps. The thermo output does not match these timesteps.
Thermo compute array is accessed out−of−range
Self−explanatory.
Thermo compute does not compute array
Self−explanatory.
Thermo compute does not compute scalar
Self−explanatory.
Thermo compute does not compute vector
Self−explanatory.
Thermo compute vector is accessed out−of−range
Self−explanatory.
Thermo custom variable cannot be indexed
Self−explanatory.
Thermo custom variable is not equal−style variable
Only equal−style variables can be output with thermodynamics, not atom−style variables.
Thermo every variable returned a bad timestep
The variable must return a timestep greater than the current timestep.
Thermo fix array is accessed out−of−range
Self−explanatory.
Thermo fix does not compute array
Self−explanatory.
Thermo fix does not compute scalar
Self−explanatory.
Thermo fix does not compute vector
Self−explanatory.
Thermo fix vector is accessed out−of−range
Self−explanatory.
Thermo keyword in variable requires lattice be defined
The xlat, ylat, zlat keywords refer to lattice properties.
Thermo keyword in variable requires thermo to use/init pe
You are using a thermo keyword in a variable that requires potential energy to be calculated, but your thermo output does not use it. Add it to your thermo output.
Thermo keyword in variable requires thermo to use/init press
You are using a thermo keyword in a variable that requires pressure to be calculated, but your thermo output does not use it. Add it to your thermo output.
Thermo keyword in variable requires thermo to use/init temp
You are using a thermo keyword in a variable that requires temperature to be calculated, but your thermo output does not use it. Add it to your thermo output.

Thermo keyword requires lattice be defined
The xlat, ylat, zlat keywords refer to lattice properties.
Thermo style does not use press
Cannot use thermo_modify to set this parameter since the thermo_style is not computing this quantity.
Thermo style does not use temp
Cannot use thermo_modify to set this parameter since the thermo_style is not computing this quantity.
Thermo_modify int format does not contain d character
Self−explanatory.
Thermo_modify pressure ID does not compute pressure
The specified compute ID does not compute pressure.
Thermo_modify temperature ID does not compute temperature
The specified compute ID does not compute temperature.
Thermo_style command before simulation box is defined
The thermo_style command cannot be used before a read_data, read_restart, or create_box command.
This variable thermo keyword cannot be used between runs
Keywords that refer to time (such as cpu, elapsed) do not make sense in between runs.
Threshhold for an atom property that isn't allocated
A dump threshhold has been requested on a quantity that is not defined by the atom style used in this simulation.
Timestep must be >= 0
Specified timestep size is invalid.
Too big a problem to use velocity create loop all
The system size must fit in a 32−bit integer to use this option.
Too big a timestep for dump dcd
The timestep must fit in a 32−bit integer to use this dump style.
Too big a timestep for dump xtc
The timestep must fit in a 32−bit integer to use this dump style.
Too few bits for lookup table
Table size specified via pair_modify command does not work with your machine's floating point representation.
Too many atom sorting bins
This is likely due to an immense simulation box that has blown up to a large size.
Too many atoms for dump dcd
The system size must fit in a 32−bit integer to use this dump style.
Too many atoms for dump xtc
The system size must fit in a 32−bit integer to use this dump style.
Too many atoms to dump sort
Cannot sort when running with more than 2^31 atoms.
Too many exponent bits for lookup table
Table size specified via pair_modify command does not work with your machine's floating point representation.
Too many groups
The maximum number of atom groups (including the "all" group) is given by MAX_GROUP in group.cpp and is 32.
Too many iterations
You must use a number of iterations that fit in a 32−bit integer for minimization.
Too many mantissa bits for lookup table
Table size specified via pair_modify command does not work with your machine's floating point representation.
Too many masses for fix shake
The fix shake command cannot list more masses than there are atom types.
Too many neighbor bins

This is likely due to an immense simulation box that has blown up to a large size.

Too many timesteps for NEB

You must use a number of timesteps that fit in a 32−bit integer for NEB.

Too many total atoms

See the setting for bigint in the src/lmptype.h file.

Too many total bits for bitmapped lookup table

Table size specified via pair_modify command is too large. Note that a value of N generates a 2^N size table.

Too many touching neighbors − boost MAXTOUCH

A granular simulation has too many neighbors touching one atom. The MAXTOUCH parameter in fix_shear_history.cpp must be set larger and LAMMPS must be re−built.

Too much per−proc info for dump

Number of local atoms times number of columns must fit in a 32−bit integer for dump.

Tree structure in joint connections

Fix poems cannot (yet) work with coupled bodies whose joints connect the bodies in a tree structure.

Triclinic box must be periodic in skewed dimensions

This is a requirement for using a non−orthogonal box. E.g. to set a non−zero xy tilt, both x and y must be periodic dimensions.

Triclinic box skew is too large

The displacement in a skewed direction must be less than half the box length in that dimension. E.g. the xy tilt must be between −half and +half of the x box length.

Tried to convert a double to int, but input_double > INT_MAX

Self−explanatory.

Two groups cannot be the same in fix spring couple

Self−explanatory.

Unbalanced quotes in input line

No matching end double quote was found following a leading double quote.

Unexpected end of data file

LAMMPS hit the end of the data file while attempting to read a section. Something is wrong with the format of the data file.

Units command after simulation box is defined

The units command cannot be used after a read_data, read_restart, or create_box command.

Universe/uloop variable count < # of partitions

A universe or uloop style variable must specify a number of values >= to the number of processor partitions.

Unknown command: %s

The command is not known to LAMMPS. Check the input script.

Unknown identifier in data file: %s

A section of the data file cannot be read by LAMMPS.

Unknown table style in angle style table

Self−explanatory.

Unknown table style in bond style table

Self−explanatory.

Unknown table style in pair_style command

Style of table is invalid for use with pair_style table command.

Unrecognized lattice type in MEAM file 1

The lattice type in an entry of the MEAM library file is not valid.

Unrecognized lattice type in MEAM file 2

The lattice type in an entry of the MEAM parameter file is not valid.

Unrecognized pair style in compute pair command

Self−explanatory.

Use of compute temp/ramp with undefined lattice

Must use lattice command with compute temp/ramp command if units option is set to lattice.

Use of displace_atoms with undefined lattice

Must use lattice command with displace_atoms command if units option is set to lattice.

Use of displace_box with undefined lattice

Must use lattice command with displace_box command if units option is set to lattice.

Use of fix ave/spatial with undefined lattice

A lattice must be defined to use fix ave/spatial with units = lattice.

Use of fix deform with undefined lattice

A lattice must be defined to use fix deform with units = lattice.

Use of fix deposit with undefined lattice

Must use lattice command with compute fix deposit command if units option is set to lattice.

Use of fix dt/reset with undefined lattice

Must use lattice command with fix dt/reset command if units option is set to lattice.

Use of fix indent with undefined lattice

The lattice command must be used to define a lattice before using the fix indent command.

Use of fix move with undefined lattice

Must use lattice command with fix move command if units option is set to lattice.

Use of fix recenter with undefined lattice

Must use lattice command with fix recenter command if units option is set to lattice.

Use of fix wall with undefined lattice

Must use lattice command with fix wall command if units option is set to lattice.

Use of region with undefined lattice

If scale = lattice (the default) for the region command, then a lattice must first be defined via the lattice command.

Use of velocity with undefined lattice

If scale = lattice (the default) for the velocity set or velocity ramp command, then a lattice must first be defined via the lattice command.

Using fix nvt/sllod with inconsistent fix deform remap option

Fix nvt/sllod requires that deforming atoms have a velocity profile provided by "remap v" as a fix deform option.

Using fix nvt/sllod with no fix deform defined

Self−explanatory.

Using fix srd with inconsistent fix deform remap option

When shearing the box in an SRD simulation, the remap v option for fix deform needs to be used.

Variable evaluation before simulation box is defined

Cannot evaluate a compute or fix or atom−based value in a variable before the simulation has been setup.

Variable for compute ti is invalid style

Self−explanatory.

Variable for dump every is invalid style

Only equal−style variables can be used.

Variable for fix adapt is invalid style

Only equal−style variables can be used.

Variable for fix addforce is invalid style

Self−explanatory.

Variable for fix aveforce is invalid style

Only equal−style variables can be used.

Variable for fix efield is invalid style

Only equal−style variables can be used.

Variable for fix indent is invalid style

Only equal−style variables can be used.

Variable for fix indent is not equal style

Only equal−style variables can be used.

Variable for fix move is invalid style
Only equal−style variables can be used.
Variable for fix setforce is invalid style
Only equal−style variables can be used.
Variable for fix wall is invalid style
Only equal−style variables can be used.
Variable for fix wall/reflect is invalid style
Only equal−style variables can be used.
Variable for fix wall/srd is invalid style
Only equal−style variables can be used.
Variable for region is invalid style
Only equal−style variables can be used.
Variable for region is not equal style
Self−explanatory.
Variable for thermo every is invalid style
Only equal−style variables can be used.
Variable for velocity set is invalid style
Only atom−style variables can be used.
Variable formula compute array is accessed out−of−range
Self−explanatory.
Variable formula compute vector is accessed out−of−range
Self−explanatory.
Variable formula fix array is accessed out−of−range
Self−explanatory.
Variable formula fix vector is accessed out−of−range
Self−explanatory.
Variable name for compute atom/molecule does not exist
Self−explanatory.
Variable name for compute reduce does not exist
Self−explanatory.
Variable name for compute ti does not exist
Self−explanatory.
Variable name for dump every does not exist
Self−explanatory.
Variable name for fix adapt does not exist
Self−explanatory.
Variable name for fix addforce does not exist
Self−explanatory.
Variable name for fix ave/atom does not exist
Self−explanatory.
Variable name for fix ave/correlate does not exist
Self−explanatory.
Variable name for fix ave/histo does not exist
Self−explanatory.
Variable name for fix ave/spatial does not exist
Self−explanatory.
Variable name for fix ave/time does not exist
Self−explanatory.
Variable name for fix aveforce does not exist
Self−explanatory.
Variable name for fix efield does not exist
Self−explanatory.

Variable name for fix indent does not exist
Self−explanatory.
Variable name for fix move does not exist
Self−explanatory.
Variable name for fix setforce does not exist
Self−explanatory.
Variable name for fix store/state does not exist
Self−explanatory.
Variable name for fix wall does not exist
Self−explanatory.
Variable name for fix wall/reflect does not exist
Self−explanatory.
Variable name for fix wall/srd does not exist
Self−explanatory.
Variable name for region does not exist
Self−explanatory.
Variable name for thermo every does not exist
Self−explanatory.
Variable name for velocity set does not exist
Self−explanatory.
Variable name must be alphanumeric or underscore characters
Self−explanatory.
Velocity command before simulation box is defined
The velocity command cannot be used before a read_data, read_restart, or create_box command.
Velocity command with no atoms existing
A velocity command has been used, but no atoms yet exist.
Velocity ramp in z for a 2d problem
Self−explanatory.
Velocity temperature ID does not compute temperature
The compute ID given to the velocity command must compute temperature.
Virial was not tallied on needed timestep
You are using a thermo keyword that requires potentials to have tallied the virial, but they didn't on this timestep. See the variable doc page for ideas on how to make this work.
Wall defined twice in fix wall command
Self−explanatory.
Wall defined twice in fix wall/reflect command
Self−explanatory.
Wall defined twice in fix wall/srd command
Self−explanatory.
Weighted neighbor list values are too big
You must have less atoms per processor to use this style neighbor list.
World variable count doesn't match # of partitions
A world−style variable must specify a number of values equal to the number of processor partitions.
Write_restart command before simulation box is defined
The write_restart command cannot be used before a read_data, read_restart, or create_box command.
Zero−length lattice orient vector
Self−explanatory.

Warnings:

All element names have been set to 'C' for dump cfg
Use the dump_modify command if you wish to override this.

Atom with molecule ID = 0 included in compute molecule group

The group used in a compute command that operates on moleclues includes atoms with no molecule ID. This is probably not what you want.

Broken bonds will not alter angles, dihedrals, or impropers

See the doc page for fix bond/break for more info on this restriction.

Building an occasional neighobr list when atoms may have moved too far

This can cause LAMMPS to crash when the neighbor list is built. The solution is to check for building the regular neighbor lists more frequently.

Compute cna/atom cutoff may be too large to find ghost atom neighbors

The neighbor cutoff used may not encompass enough ghost atoms to perform this operation correctly.

Computing temperature of portions of rigid bodies

The group defined by the temperature compute does not encompass all the atoms in one or more rigid bodies, so the change in degrees−of−freedom for the atoms in those partial rigid bodies will not be accounted for.

Created bonds will not create angles, dihedrals, or impropers

See the doc page for fix bond/create for more info on this restriction.

Dihedral problem: %d %d %d %d %d %d

Conformation of the 4 listed dihedral atoms is extreme; you may want to check your simulation geometry.

Dump dcd/xtc timestamp may be wrong with fix dt/reset

If the fix changes the timestep, the dump dcd file will not reflect the change.

FENE bond too long: %d %d %d %g

A FENE bond has stretched dangerously far. It's interaction strength will be truncated to attempt to prevent the bond from blowing up.

FENE bond too long: %d %g

A FENE bond has stretched dangerously far. It's interaction strength will be truncated to attempt to prevent the bond from blowing up.

Fix SRD walls overlap but fix srd overlap not set

You likely want to set this in your input script.

Fix bond/swap will ignore defined angles

See the doc page for fix bond/swap for more info on this restriction.

Fix move does not update angular momentum

Atoms store this quantity, but fix move does not (yet) update it.

Fix move does not update quaternions

Atoms store this quantity, but fix move does not (yet) update it.

Fix recenter should come after all other integration fixes

Other fixes may change the position of the center−of−mass, so fix recenter should come last.

Fix srd SRD moves may trigger frequent reneighboring

This is because the SRD particles may move long distances.

Fix srd grid size > 1/4 of big particle diameter

This may cause accuracy problems.

Fix srd no−slip wall collisions with bin shifting

This is an inconsistent setting in your input script.

Fix srd particle moved outside valid domain

This may indicate a problem with your simulation parameters.

Fix srd particles may move > big particle diameter

This may cause accuracy problems.

Fix srd viscosity < 0.0 due to low SRD density

This may cause accuracy problems.

Fix thermal/conductivity comes before fix ave/spatial

The order of these 2 fixes in your input script is such that fix thermal/conductivity comes first. If you are using fix ave/spatial to measure the temperature profile induced by fix viscosity, then this may cause a glitch in the profile since you are averaging immediately after swaps have occurred. Flipping the order of

the 2 fixes typically helps.

Fix viscosity comes before fix ave/spatial

The order of these 2 fixes in your input script is such that fix viscosity comes first. If you are using fix ave/spatial to measure the velocity profile induced by fix viscosity, then this may cause a glitch in the profile since you are averaging immediately after swaps have occurred. Flipping the order of the 2 fixes typically helps.

Group for fix_modify temp != fix group

The fix_modify command is specifying a temperature computation that computes a temperature on a different group of atoms than the fix itself operates on. This is probably not what you want to do.

Improper problem: %d %d %d %d %d %d

Conformation of the 4 listed improper atoms is extreme; you may want to check your simulation geometry.

Kspace_modify slab param < 2.0 may cause unphysical behavior

The kspace_modify slab parameter should be larger to insure periodic grids padded with empty space do not overlap.

Less insertions than requested

Less atom insertions occurred on this timestep due to the fix pour command than were scheduled. This is probably because there were too many overlaps detected.

Lost atoms: original %.15g current %.15g

A thermodynamic computation has detected lost atoms.

Mismatch between velocity and compute groups

The temperature computation used by the velocity command will not be on the same group of atoms that velocities are being set for.

More than one compute centro/atom

It is not efficient to use compute centro/atom more than once.

More than one compute cluster/atom

It is not efficient to use compute cluster/atom more than once.

More than one compute cna/atom defined

It is not efficient to use compute cna/atom more than once.

More than one compute coord/atom

It is not efficient to use compute coord/atom more than once.

More than one compute damage/atom

It is not efficient to use compute ke/atom more than once.

More than one compute ke/atom

It is not efficient to use compute ke/atom more than once.

More than one fix poems

It is not efficient to use fix poems more than once.

More than one fix rigid

It is not efficient to use fix rigid more than once.

New thermo_style command, previous thermo_modify settings will be lost

If a thermo_style command is used after a thermo_modify command, the settings changed by the thermo_modify command will be reset to their default values. This is because the thermo_modify commmand acts on the currently defined thermo style, and a thermo_style command creates a new style.

No fixes defined, atoms won't move

If you are not using a fix like nve, nvt, npt then atom velocities and coordinates will not be updated during timestepping.

No joints between rigid bodies, use fix rigid instead

The bodies defined by fix poems are not connected by joints. POEMS will integrate the body motion, but it would be more efficient to use fix rigid.

Not using real units with pair reax

This is most likely an error, unless you have created your own ReaxFF parameter file in a different set of units.

One or more atoms are time integrated more than once
: This is probably an error since you typically do not want to advance the positions or velocities of an atom more than once per timestep.

One or more compute molecules has atoms not in group
: The group used in a compute command that operates on moleclues does not include all the atoms in some molecules. This is probably not what you want.

One or more respa levels compute no forces
: This is computationally inefficient.

Pair COMB charge %.10f with force %.10f hit max barrier
: Something is possibly wrong with your model.

Pair COMB charge %.10f with force %.10f hit min barrier
: Something is possibly wrong with your model.

Pair dsmc: num_of_collisions > number_of_A
: Collision model in DSMC is breaking down.

Pair dsmc: num_of_collisions > number_of_B
: Collision model in DSMC is breaking down.

Particle deposition was unsuccessful
: The fix deposit command was not able to insert as many atoms as needed. The requested volume fraction may be too high, or other atoms may be in the insertion region.

Reducing PPPM order b/c stencil extends beyond neighbor processor
: LAMMPS is attempting this in order to allow the simulation to run. It should not effect the PPPM accuracy.

Replacing a fix, but new group != old group
: The ID and style of a fix match for a fix you are changing with a fix command, but the new group you are specifying does not match the old group.

Replicating in a non−periodic dimension
: The parameters for a replicate command will cause a non−periodic dimension to be replicated; this may cause unwanted behavior.

Resetting reneighboring criteria during PRD
: A PRD simulation requires that neigh_modify settings be delay = 0, every = 1, check = yes. Since these settings were not in place, LAMMPS changed them and will restore them to their original values after the PRD simulation.

Resetting reneighboring criteria during TAD
: A TAD simulation requires that neigh_modify settings be delay = 0, every = 1, check = yes. Since these settings were not in place, LAMMPS changed them and will restore them to their original values after the PRD simulation.

Resetting reneighboring criteria during minimization
: Minimization requires that neigh_modify settings be delay = 0, every = 1, check = yes. Since these settings were not in place, LAMMPS changed them and will restore them to their original values after the minimization.

Restart file used different # of processors
: The restart file was written out by a LAMMPS simulation running on a different number of processors. Due to round−off, the trajectories of your restarted simulation may diverge a little more quickly than if you ran on the same # of processors.

Restart file used different 3d processor grid
: The restart file was written out by a LAMMPS simulation running on a different 3d grid of processors. Due to round−off, the trajectories of your restarted simulation may diverge a little more quickly than if you ran on the same # of processors.

Restart file used different boundary settings, using restart file values
: Your input script cannot change these restart file settings.

Restart file used different newton bond setting, using restart file value
: The restart file value will override the setting in the input script.

Restart file used different newton pair setting, using input script value
The input script value will override the setting in the restart file.
Restart file version does not match LAMMPS version
This may cause problems when reading the restart file.
Running PRD with only one replica
This is allowed, but you will get no parallel speed−up.
SRD bin shifting turned on due to small lamda
This is done to try to preserve accuracy.
SRD bin size for fix srd differs from user request
Check if the new bin size is acceptable.
SRD bins for fix srd are not cubic enough
Check if the bin shape is acceptable.
SRD particle %d started inside big particle %d on step %d bounce %d
This may not be a problem, but indicates one or more SRD particles are being left inside solute particles.
Shake determinant < 0.0
The determinant of the quadratic equation being solved for a single cluster specified by the fix shake command is numerically suspect. LAMMPS will set it to 0.0 and continue.
Should not allow rigid bodies to bounce off relecting walls
LAMMPS allows this, but their dynamics are not computed correctly.
System is not charge neutral, net charge = %g
The total charge on all atoms on the system is not 0.0, which is not valid for Ewald or PPPM.
Table inner cutoff >= outer cutoff
You specified an inner cutoff for a Coulombic table that is longer than the global cutoff. Probably not what you wanted.
Temperature for MSST is not for group all
User−assigned temperature to MSST fix does not compute temperature for all atoms. Since MSST computes a global pressure, the kinetic energy contribution from the temperature is assumed to also be for all atoms. Thus the pressure used by MSST could be inaccurate.
Temperature for NPT is not for group all
User−assigned temperature to NPT fix does not compute temperature for all atoms. Since NPT computes a global pressure, the kinetic energy contribution from the temperature is assumed to also be for all atoms. Thus the pressure used by NPT could be inaccurate.
Temperature for fix modify is not for group all
The temperature compute is being used with a pressure calculation which does operate on group all, so this may be inconsistent.
Temperature for thermo pressure is not for group all
User−assigned temperature to thermo via the thermo_modify command does not compute temperature for all atoms. Since thermo computes a global pressure, the kinetic energy contribution from the temperature is assumed to also be for all atoms. Thus the pressure printed by thermo could be inaccurate.
Too many common neighbors in CNA %d times
More than the maximum # of neighbors was found multiple times. This was unexpected.
Too many inner timesteps in fix ttm
Self−explanatory.
Too many neighbors in CNA for %d atoms
More than the maximum # of neighbors was found multiple times. This was unexpected.
Use special bonds = 0,1,1 with bond style fene/expand
Most FENE models need this setting for the special_bonds command.
Use special bonds = 0,1,1 with bond style fene
Most FENE models need this setting for the special_bonds command.
Using compute temp/deform with inconsistent fix deform remap option
Fix nvt/sllod assumes deforming atoms have a velocity profile provided by "remap v" or "remap none" as a fix deform option.

Using compute temp/deform with no fix deform defined

This is probably an error, since it makes little sense to use compute temp/deform in this case.

Using pair tail corrections with nonperiodic system

This is probably a bogus thing to do, since tail corrections are computed by integrating the density of a periodic system out to infinity.

13. Future and history

This section lists features we are planning to add to LAMMPS, features of previous versions of LAMMPS, and features of other parallel molecular dynamics codes I've distributed.

13.1 Coming attractions
13.2 Past versions

13.1 Coming attractions

The current version of LAMMPS incorporates nearly all the features from previous parallel MD codes developed at Sandia. These include earlier versions of LAMMPS itself, Warp and ParaDyn for metals, and GranFlow for granular materials.

These are new features we'd like to eventually add to LAMMPS. Some are being worked on; some haven't been implemented because of lack of time or interest; others are just a lot of work! See this page on the LAMMPS WWW site for more details.

- Coupling to finite elements for streess−strain
- New ReaxFF implementation
- Nudged elastic band
- Temperature accelerated dynamics
- Triangulated particles
- Stochastic rotation dynamics
- Stokesian dynamics via fast lubrication dynamics
- Long−range point−dipole solver
- Per−atom energy and stress for long−range Coulombics
- Long−range Coulombics via Ewald and PPPM for triclinic boxes
- Metadynamics
- Direct Simulation Monte Carlo − DSMC

13.2 Past versions

LAMMPS development began in the mid 1990s under a cooperative research &development agreement (CRADA) between two DOE labs (Sandia and LLNL) and 3 companies (Cray, Bristol Myers Squibb, and Dupont). Soon after the CRADA ended, a final F77 version of the code, LAMMPS 99, was released. As development of LAMMPS continued at Sandia, the memory management in the code was converted to F90; a final F90 version was released as LAMMPS 2001.

The current LAMMPS is a rewrite in C++ and was first publicly released in 2004. It includes many new features, including features from other parallel molecular dynamics codes written at Sandia, namely ParaDyn, Warp, and GranFlow. ParaDyn is a parallel implementation of the popular serial DYNAMO code developed by Stephen Foiles and Murray Daw for their embedded atom method (EAM) metal potentials. ParaDyn uses atom− and force−decomposition algorithms to run in parallel. Warp is also a parallel implementation of the EAM potentials designed for large problems, with boundary conditions specific to shearing solids in varying geometries. GranFlow is a granular materials code with potentials and boundary conditions peculiar to granular systems. All of these codes (except ParaDyn) use spatial−decomposition techniques for their parallelism.

These older codes are available for download from the LAMMPS WWW site, except for Warp &GranFlow which were primarily used internally. A brief listing of their features is given here.

LAMMPS 2001

- F90 + MPI
- dynamic memory
- spatial−decomposition parallelism
- NVE, NVT, NPT, NPH, rRESPA integrators
- LJ and Coulombic pairwise force fields
- all−atom, united−atom, bead−spring polymer force fields
- CHARMM−compatible force fields
- class 2 force fields
- 3d/2d Ewald &PPPM
- various force and temperature constraints
- SHAKE
- Hessian−free truncated−Newton minimizer
- user−defined diagnostics

LAMMPS 99

- F77 + MPI
- static memory allocation
- spatial−decomposition parallelism
- most of the LAMMPS 2001 features with a few exceptions
- no 2d Ewald &PPPM
- molecular force fields are missing a few CHARMM terms
- no SHAKE

Warp

- F90 + MPI
- spatial−decomposition parallelism
- embedded atom method (EAM) metal potentials + LJ
- lattice and grain−boundary atom creation
- NVE, NVT integrators
- boundary conditions for applying shear stresses
- temperature controls for actively sheared systems
- per−atom energy and centro−symmetry computation and output

ParaDyn

- F77 + MPI
- atom− and force−decomposition parallelism
- embedded atom method (EAM) metal potentials
- lattice atom creation
- NVE, NVT, NPT integrators
- all serial DYNAMO features for controls and constraints

GranFlow

- F90 + MPI
- spatial−decomposition parallelism
- frictional granular potentials
- NVE integrator

- boundary conditions for granular flow and packing and walls
- particle insertion

angle_style charmm command

Syntax:

```
angle_style charmm
```

Examples:

```
angle_style charmm
angle_coeff 1 300.0 107.0 50.0 3.0
```

Description:

The *charmm* angle style uses the potential

$$E = K(\theta - \theta_0)^2 + K_{UB}(r - r_{UB})^2$$

with an additional Urey_Bradley term based on the distance *r* between the 1st and 3rd atoms in the angle. K, theta0, Kub, and Rub are coefficients defined for each angle type.

See (MacKerell) for a description of the CHARMM force field.

The following coefficients must be defined for each angle type via the angle_coeff command as in the example above, or in the data file or restart files read by the read_data or read_restart commands:

- K (energy/radian^2)
- theta0 (degrees)
- K_ub (energy/distance^2)
- r_ub (distance)

Theta0 is specified in degrees, but LAMMPS converts it to radians internally; hence the units of K are in energy/radian^2.

Restrictions:

This angle style can only be used if LAMMPS was built with the "molecular" package (which it is by default). See the Making LAMMPS section for more info on packages.

Related commands:

angle_coeff

Default: none

(MacKerell) MacKerell, Bashford, Bellott, Dunbrack, Evanseck, Field, Fischer, Gao, Guo, Ha, et al, J Phys Chem, 102, 3586 (1998).

angle_style class2 command

Syntax:

```
angle_style class2
```

Examples:

```
angle_style class2
angle_coeff * 75.0
angle_coeff 1 bb 10.5872 1.0119 1.5228
angle_coeff * ba 3.6551 24.895 1.0119 1.5228
```

Description:

The *class2* angle style uses the potential

$$
\begin{aligned}
E &= E_a + E_{bb} + E_{ba} \\
E_a &= K_2(\theta - \theta_0)^2 + K_3(\theta - \theta_0)^3 + K_4(\theta - \theta_0)^4 \\
E_{bb} &= M(r_{ij} - r_1)(r_{jk} - r_2) \\
E_{ba} &= N_1(r_{ij} - r_1)(\theta - \theta_0) + N_2(r_{jk} - r_2)(\theta - \theta_0)
\end{aligned}
$$

where Ea is the angle term, Ebb is a bond−bond term, and Eba is a bond−angle term. Theta0 is the equilibrium angle and r1 and r2 are the equilibrium bond lengths.

See (Sun) for a description of the COMPASS class2 force field.

Coefficients for the Ea, Ebb, and Eba formulas must be defined for each angle type via the bond_coeff command as in the example above, or in the data file or restart files read by the read_data or read_restart commands.

These are the 4 coefficients for the Ea formula:

- theta0 (degrees)
- K2 (energy/radian^2)
- K3 (energy/radian^3)
- K4 (energy/radian^4)

Theta0 is specified in degrees, but LAMMPS converts it to radians internally; hence the units of the various K are in per−radian.

For the Ebb formula, each line in a bond_coeff command in the input script lists 4 coefficients, the first of which is "bb" to indicate they are BondBond coefficients. In a data file, these coefficients should be listed under a "BondBond Coeffs" heading and you must leave out the "bb", i.e. only list 3 coefficients after the angle type.

- bb
- M (energy/distance^2)
- r1 (distance)
- r2 (distance)

For the Eba formula, each line in a bond_coeff command in the input script lists 5 coefficients, the first of which is "ba" to indicate they are BondAngle coefficients. In a data file, these coefficients should be listed under a "BondAngle Coeffs" heading and you must leave out the "ba", i.e. only list 4 coefficients after the angle type.

- ba
- N1 (energy/distance^2)
- N2 (energy/distance^2)
- r1 (distance)
- r2 (distance)

The theta0 value in the Eba formula is not specified, since it is the same value from the Ea formula.

Restrictions:

This angle style can only be used if LAMMPS was built with the "class2" package. See the Making LAMMPS section for more info on packages.

Related commands:

angle_coeff

Default: none

(Sun) Sun, J Phys Chem B 102, 7338−7364 (1998).

angle_style cg/cmm command

Syntax:

```
angle_style cg/cmm
```

Examples:

```
angle_style cg/cmm
angle_coeff 1 300.0 107.0 lj9_6 0.4491 3.7130
```

Description:

The *cg/cmm* angle style is a combination of the harmonic angle potential,

$$E = K(\theta - \theta_0)^2$$

where theta0 is the equilibrium value of the angle and K a prefactor, with the *repulsive* part of the non−bonded *cg/cmm* pair style between the atoms 1 and 3. This angle potential is intended for coarse grained MD simulations with the CMM parametrization using the pair_style cg/cmm. Relative to the pair_style *cg/cmm*, however, the energy is shifted by *epsilon*, to avoid sudden jumps. Note that the usual 1/2 factor is included in K.

The following coefficients must be defined for each angle type via the angle_coeff command as in the example above. As with other CMM coarse grained parameters, they cannot be set in the data file, but can be restored from restarts via the read_restart command:

- K (energy/radian^2)
- theta0 (degrees)
- cg_type (string, one of lj9_6, lj12_4, lj12_6)
- epsilon (energy units)
- sigma (distance units)

Theta0 is specified in degrees, but LAMMPS converts it to radians internally; hence the units of K are in energy/radian^2.

Restrictions:

This angle style can only be used if LAMMPS was built with the "user−cg−cmm" package. See the Making LAMMPS section for more info on packages.

Related commands:

angle_coeff, angle_style harmonic, pair_style cg/cmm

Default: none

angle_coeff command

Syntax:

```
angle_coeff N args
```

- N = angle type (see asterisk form below)
- args = coefficients for one or more angle types

Examples:

```
angle_coeff 1 300.0 107.0
angle_coeff * 5.0
angle_coeff 2*10 5.0
```

Description:

Specify the angle force field coefficients for one or more angle types. The number and meaning of the coefficients depends on the angle style. Angle coefficients can also be set in the data file read by the read_data command or in a restart file.

N can be specified in one of two ways. An explicit numeric value can be used, as in the 1st example above. Or a wild−card asterisk can be used to set the coefficients for multiple angle types. This takes the form "*" or "*n" or "n*" or "m*n". If N = the number of angle types, then an asterisk with no numeric values means all types from 1 to N. A leading asterisk means all types from 1 to n (inclusive). A trailing asterisk means all types from n to N (inclusive). A middle asterisk means all types from m to n (inclusive).

Note that using an angle_coeff command can override a previous setting for the same angle type. For example, these commands set the coeffs for all angle types, then overwrite the coeffs for just angle type 2:

```
angle_coeff * 200.0 107.0 1.2
angle_coeff 2 50.0 107.0
```

A line in a data file that specifies angle coefficients uses the exact same format as the arguments of the angle_coeff command in an input script, except that wild−card asterisks should not be used since coefficients for all N types must be listed in the file. For example, under the "Angle Coeffs" section of a data file, the line that corresponds to the 1st example above would be listed as

```
1 300.0 107.0
```

The angle_style class2 is an exception to this rule, in that an additional argument is used in the input script to allow specification of the cross−term coefficients. See its doc page for details.

Here is an alphabetic list of angle styles defined in LAMMPS. Click on the style to display the formula it computes and coefficients specified by the associated angle_coeff command:

- angle_style none − turn off angle interactions
- angle_style hybrid − define multiple styles of angle interactions

- angle_style charmm − CHARMM angle
- angle_style class2 − COMPASS (class 2) angle

- angle_style cosine − cosine angle potential
- angle_style cosine/delta − difference of cosines angle potential
- angle_style cosine/periodic − DREIDING angle
- angle_style cosine/squared − cosine squared angle potential
- angle_style harmonic − harmonic angle
- angle_style table − tabulated by angle

There are also additional angle styles submitted by users which are included in the LAMMPS distribution. The list of these with links to the individual styles are given in the angle section of this page.

Restrictions:

This command must come after the simulation box is defined by a read_data, read_restart, or create_box command.

An angle style must be defined before any angle coefficients are set, either in the input script or in a data file.

Related commands:

angle_style

Default: none

angle_style cosine command

Syntax:

```
angle_style cosine
```

Examples:

```
angle_style cosine
angle_coeff * 75.0
```

Description:

The *cosine* angle style uses the potential

$$E = K[1 + \cos(\theta)]$$

where K is defined for each angle type.

The following coefficients must be defined for each angle type via the angle_coeff command as in the example above, or in the data file or restart files read by the read_data or read_restart commands:

- K (energy)

Restrictions:

This angle style can only be used if LAMMPS was built with the "molecular" package (which it is by default). See the Making LAMMPS section for more info on packages.

Related commands:

angle_coeff

Default: none

angle_style cosine/delta command

Syntax:

```
angle_style cosine/delta
```

Examples:

```
angle_style cosine/delta
angle_coeff 2*4 75.0 100.0
```

Description:

The *cosine/delta* angle style uses the potential

$$E = K[\cos(\theta) - \cos(\theta_0)]$$

where theta0 is the equilibrium value of the angle, and K is a prefactor. Note that the usual 1/2 factor is included in K.

The following coefficients must be defined for each angle type via the angle_coeff command as in the example above, or in the data file or restart files read by the read_data or read_restart commands:

- K (energy)
- theta0 (degrees)

Theta0 is specified in degrees, but LAMMPS converts it to radians internally.

Restrictions:

This angle style can only be used if LAMMPS was built with the "molecular" package (which it is by default). See the Making LAMMPS section for more info on packages.

Related commands:

angle_coeff, angle_style cosine/squared

Default: none

angle_style cosine/periodic command

Syntax:

```
angle_style cosine/periodic
```

Examples:

```
angle_style cosine/periodic
angle_coeff * 75.0 1 6
```

Description:

The *cosine/periodic* angle style uses the following potential, which is commonly used in the DREIDING force field, particularly for organometallic systems where *n* = 4 might be used for an octahedral complex and *n* = 3 might be used for a trigonal center:

$$E = C\left[1 - B(-1)^n cos\left(n\theta\right)\right]$$

where C, B and n are coefficients defined for each angle type.

See (Mayo) for a description of the DREIDING force field

The following coefficients must be defined for each angle type via the angle_coeff command as in the example above, or in the data file or restart files read by the read_data or read_restart commands:

- C (energy)
- B = 1 or −1
- n = 1, 2, 3, 4, 5 or 6 for periodicity

Note that the prefactor C is specified and not the overall force constant K = C / n^2. When B = 1, it leads to a minimum for the linear geometry. When B = −1, it leads to a maximum for the linear geometry.

Restrictions:

This angle style can only be used if LAMMPS was built with the "molecular" package (which it is by default). See the Making LAMMPS section for more info on packages.

Related commands:

angle_coeff

Default: none

(Mayo) Mayo, Olfason, Goddard III, J Phys Chem, 94, 8897−8909 (1990).

angle_style cosine/shift command

Syntax:

```
angle_style cosine/shift
```

Examples:

```
angle_style cosine/shift
angle_coeff * 10.0 45.0
```

Description:

The *cosine/shift* angle style uses the potential

$$E = -\frac{Umin}{2}[1 + Cos(\theta - \theta_0)]$$

where theta0 is the equilibrium angle. The potential is bounded between −Umin and zero. In the neighborhood of the minimum E=− Umin + Umin/4(theta−theta0)^2 hence the spring constant is umin/2.

The following coefficients must be defined for each angle type via the angle_coeff command as in the example above, or in the data file or restart files read by the read_data or read_restart commands:

- umin (energy)
- theta (angle)

Restrictions:

This angle style can only be used if LAMMPS was built with the "user−misc" package. See the Making LAMMPS section for more info on packages.

Related commands:

angle_coeff, angle_cosineshiftexp

Default: none

angle_style cosine/shift/exp command

Syntax:

```
angle_style cosine/shift/exp
```

Examples:

```
angle_style cosine/shift/exp
angle_coeff * 10.0 45.0 2.0
```

Description:

The *cosine/shift/exp* angle style uses the potential

$$E = -U_{min}\frac{e^{-aU(\theta,\theta_0)}-1}{e^a-1} \quad \text{with} \quad U(\theta,\theta_0) = -0.5\left(1+\cos(\theta-\theta_0)\right)$$

where Umin, theta, and a are defined for each angle type.

The potential is bounded between [−Umin:0] and the minimum is located at the angle theta0. The a parameter can be both positive or negative and is used to control the spring constant at the equilibrium.

The spring constant is given by k = A exp(A) Umin / [2 (Exp(a)−1)]. For a > 3, k/Umin = a/2 to better than 5% relative error. For negative values of the a parameter, the spring constant is essentially zero, and anharmonic terms takes over. The potential is furthermore well behaved in the limit a −> 0, where it has been implemented to linear order in a for a < 0.001. In this limit the potential reduces to the cosineshifted potential.

The following coefficients must be defined for each angle type via the angle_coeff command as in the example above, or in the data file or restart files read by the read_data or read_restart commands:

- umin (energy)
- theta (angle)
- A (real number)

Restrictions:

This angle style can only be used if LAMMPS was built with the "user−misc" package. See the Making LAMMPS section for more info on packages.

Related commands:

angle_coeff, angle_cosineshift, dihedral_cosineshift

Default: none

angle_style cosine/squared command

Syntax:

```
angle_style cosine/squared
```

Examples:

```
angle_style cosine/squared
angle_coeff 2*4 75.0 100.0
```

Description:

The *cosine/squared* angle style uses the potential

$$E = K[\cos(\theta) - \cos(\theta_0)]^2$$

where theta0 is the equilibrium value of the angle, and K is a prefactor. Note that the usual 1/2 factor is included in K.

The following coefficients must be defined for each angle type via the angle_coeff command as in the example above, or in the data file or restart files read by the read_data or read_restart commands:

- K (energy)
- theta0 (degrees)

Theta0 is specified in degrees, but LAMMPS converts it to radians internally.

Restrictions:

This angle style can only be used if LAMMPS was built with the "molecular" package (which it is by default). See the Making LAMMPS section for more info on packages.

Related commands:

angle_coeff

Default: none

angle_style harmonic command

Syntax:

```
angle_style harmonic
```

Examples:

```
angle_style harmonic
angle_coeff 1 300.0 107.0
```

Description:

The *harmonic* angle style uses the potential

$$E = K(\theta - \theta_0)^2$$

where theta0 is the equilibrium value of the angle, and K is a prefactor. Note that the usual 1/2 factor is included in K.

The following coefficients must be defined for each angle type via the angle_coeff command as in the example above, or in the data file or restart files read by the read_data or read_restart commands:

- K (energy/radian^2)
- theta0 (degrees)

Theta0 is specified in degrees, but LAMMPS converts it to radians internally; hence the units of K are in energy/radian^2.

Restrictions: none

This angle style can only be used if LAMMPS was built with the "molecular" package (which it is by default). See the Making LAMMPS section for more info on packages.

Related commands:

angle_coeff

Default: none

angle_style hybrid command

Syntax:

```
angle_style hybrid style1 style2 ...
```

- style1,style2 = list of one or more angle styles

Examples:

```
angle_style hybrid harmonic cosine
angle_coeff 1 harmonic 80.0 30.0
angle_coeff 2* cosine 50.0
```

Description:

The *hybrid* style enables the use of multiple angle styles in one simulation. An angle style is assigned to each angle type. For example, angles in a polymer flow (of angle type 1) could be computed with a *harmonic* potential and angles in the wall boundary (of angle type 2) could be computed with a *cosine* potential. The assignment of angle type to style is made via the angle_coeff command or in the data file.

In the angle_coeff commands, the name of an angle style must be added after the angle type, with the remaining coefficients being those appropriate to that style. In the example above, the 2 angle_coeff commands set angles of angle type 1 to be computed with a *harmonic* potential with coefficients 80.0, 30.0 for K, theta0. All other angle types (2−N) are computed with a *cosine* potential with coefficient 50.0 for K.

If angle coefficients are specified in the data file read via the read_data command, then the same rule applies. E.g. "harmonic" or "cosine", must be added after the angle type, for each line in the "Angle Coeffs" section, e.g.

```
Angle Coeffs

1 harmonic 80.0 30.0
2 cosine 50.0
...
```

If *class2* is one of the angle hybrid styles, the same rule holds for specifying additional BondBond (and BondAngle) coefficients either via the input script or in the data file. I.e. *class2* must be added to each line after the angle type. For lines in the BondBond (or BondAngle) section of the data file for angle types that are not *class2*, you must use an angle style of *skip* as a placeholder, e.g.

```
BondBond Coeffs

1 skip
2 class2 3.6512 1.0119 1.0119
...
```

Note that it is not necessary to use the angle style *skip* in the input script, since BondBond (or BondAngle) coefficients need not be specified at all for angle types that are not *class2*.

An angle style of *none* with no additional coefficients can be used in place of an angle style, either in a input script angle_coeff command or in the data file, if you desire to turn off interactions for specific angle types.

Restrictions:

This angle style can only be used if LAMMPS was built with the "molecular" package (which it is by default). See the Making LAMMPS section for more info on packages.

Unlike other angle styles, the hybrid angle style does not store angle coefficient info for individual sub−styles in a binary restart files. Thus when retarting a simulation from a restart file, you need to re−specify angle_coeff commands.

Related commands:

angle_coeff

Default: none

angle_style none command

Syntax:

```
angle_style none
```

Examples:

```
angle_style none
```

Description:

Using an angle style of none means angle forces are not computed, even if triplets of angle atoms were listed in the data file read by the read_data command.

Restrictions: none

Related commands: none

Default: none

angle_style command

Syntax:

```
angle_style style
```

- style = *none* or *hybrid* or *charmm* or *class2* or *cosine* or *cosine/squared* or *harmonic*

Examples:

```
angle_style harmonic
angle_style charmm
angle_style hybrid harmonic cosine
```

Description:

Set the formula(s) LAMMPS uses to compute angle interactions between triplets of atoms, which remain in force for the duration of the simulation. The list of angle triplets is read in by a read_data or read_restart command from a data or restart file.

Hybrid models where angles are computed using different angle potentials can be setup using the *hybrid* angle style.

The coefficients associated with a angle style can be specified in a data or restart file or via the angle_coeff command.

All angle potentials store their coefficient data in binary restart files which means angle_style and angle_coeff commands do not need to be re−specified in an input script that restarts a simulation. See the read_restart command for details on how to do this. The one exception is that angle_style *hybrid* only stores the list of sub−styles in the restart file; angle coefficients need to be re−specified.

IMPORTANT NOTE: When both an angle and pair style is defined, the special_bonds command often needs to be used to turn off (or weight) the pairwise interaction that would otherwise exist between 3 bonded atoms.

In the formulas listed for each angle style, *theta* is the angle between the 3 atoms in the angle.

Here is an alphabetic list of angle styles defined in LAMMPS. Click on the style to display the formula it computes and coefficients specified by the associated angle_coeff command:

- angle_style none − turn off angle interactions
- angle_style hybrid − define multiple styles of angle interactions

- angle_style charmm − CHARMM angle
- angle_style class2 − COMPASS (class 2) angle
- angle_style cosine − cosine angle potential
- angle_style cosine/delta − difference of cosines angle potential
- angle_style cosine/periodic − DREIDING angle
- angle_style cosine/squared − cosine squared angle potential
- angle_style harmonic − harmonic angle
- angle_style table − tabulated by angle

There are also additional angle styles submitted by users which are included in the LAMMPS distribution. The list of these with links to the individual styles are given in the angle section of this page.

Restrictions:

Angle styles can only be set for atom_styles that allow angles to be defined.

Most angle styles are part of the "molecular" package. They are only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info on packages. The doc pages for individual bond potentials tell if it is part of a package.

Related commands:

angle_coeff

Default:

```
angle_style none
```

angle_style table command

Syntax:

```
angle_style table style N
```

- style = *linear* or *spline* = method of interpolation
- N = use N values in table

Examples:

```
angle_style table linear 1000
angle_coeff 3 file.table ENTRY1
```

Description:

Style *table* creates interpolation tables of length *N* from angle potential and derivative values listed in a file(s) as a function of angle The files are read by the angle_coeff command.

The interpolation tables are created by fitting cubic splines to the file values and interpolating energy and derivative values at each of *N* angles. During a simulation, these tables are used to interpolate energy and force values on individual atoms as needed. The interpolation is done in one of 2 styles: *linear* or *spline*.

For the *linear* style, the angle is used to find 2 surrounding table values from which an energy or its derivative is computed by linear interpolation.

For the *spline* style, a cubic spline coefficients are computed and stored at each of the *N* values in the table. The angle is used to find the appropriate set of coefficients which are used to evaluate a cubic polynomial which computes the energy or derivative.

The following coefficients must be defined for each angle type via the angle_coeff command as in the example above.

- filename
- keyword

The filename specifies a file containing tabulated energy and derivative values. The keyword specifies a section of the file. The format of this file is described below.

The format of a tabulated file is as follows (without the parenthesized comments):

```
# Angle potential for harmonic (one or more comment or blank lines)

HAM                           (keyword is the first text on line)
N 181 FP 0 0 EQ 90.0          (N, FP, EQ parameters)
                              (blank line)
N 181 FP 0 0                  (N, FP parameters)
1 0.0 200.5 2.5               (index, angle, energy, derivative)
2 1.0 198.0 2.5
...
181 180.0 0.0 0.0
```

A section begins with a non−blank line whose 1st character is not a "#"; blank lines or lines starting with "#" can be used as comments between sections. The first line begins with a keyword which identifies the section. The line can contain additional text, but the initial text must match the argument specified in the angle_coeff command. The next line lists (in any order) one or more parameters for the table. Each parameter is a keyword followed by one or more numeric values.

The parameter "N" is required and its value is the number of table entries that follow. Note that this may be different than the *N* specified in the angle_style table command. Let Ntable = *N* in the angle_style command, and Nfile = "N" in the tabulated file. What LAMMPS does is a preliminary interpolation by creating splines using the Nfile tabulated values as nodal points. It uses these to interpolate as needed to generate energy and derivative values at Ntable different points. The resulting tables of length Ntable are then used as described above, when computing energy and force for individual angles and their atoms. This means that if you want the interpolation tables of length Ntable to match exactly what is in the tabulated file (with effectively no preliminary interpolation), you should set Ntable = Nfile.

The "FP" parameter is optional. If used, it is followed by two values fplo and fphi, which are the 2nd derivatives at the innermost and outermost angle settings. These values are needed by the spline construction routines. If not specified by the "FP" parameter, they are estimated (less accurately) by the first two and last two derivative values in the table.

The "EQ" parameter is also optional. If used, it is followed by a the equilibrium angle value, which is used, for example, by the fix shake command. If not used, the equilibrium angle is set to 180.0.

Following a blank line, the next N lines list the tabulated values. On each line, the 1st value is the index from 1 to N, the 2nd value is the angle value (in degrees), the 3rd value is the energy (in energy units), and the 4th is −dE/d(theta) (also in energy units). The 3rd term is the energy of the 3−atom configuration for the specified angle. The last term is the derivative of the energy with respect to the angle (in degrees, not radians). Thus the units of the last term are still energy, not force. The angle values must increase from one line to the next. The angle values must also begin with 0.0 and end with 180.0, i.e. span the full range of possible angles.

Note that one file can contain many sections, each with a tabulated potential. LAMMPS reads the file section by section until it finds one that matches the specified keyword.

Restrictions:

This angle style can only be used if LAMMPS was built with the "molecular" package (which it is by default). See the Making LAMMPS section for more info on packages.

Related commands:

angle_coeff

Default: none

atom_modify command

Syntax:

```
atom_modify keyword values ...
```

- one or more keyword/value pairs may be appended
- keyword = *map* or *first* or *sort*

```
  map value = array or hash
    first value = group-ID = group whose atoms will appear first in internal atom lists
    sort values = Nfreq binsize
      Nfreq = sort atoms spatially every this many time steps
      binsize = bin size for spatial sorting (distance units)
```

Examples:

```
atom_modify map hash
atom_modify map array sort 10000 2.0
atom_modify first colloid
```

Description:

Modify properties of the atom style selected within LAMMPS.

The *map* keyword determines how atom ID lookup is done for molecular problems. Lookups are performed by bond (angle, etc) routines in LAMMPS to find the local atom index associated with a global atom ID. When the *array* value is used, each processor stores a lookup table of length N, where N is the total # of atoms in the system. This is the fastest method for most simulations, but a processor can run out of memory to store the table for very large simulations. The *hash* value uses a hash table to perform the lookups. This method can be slightly slower than the *array* method, but its memory cost is proportional to N/P on each processor, where P is the total number of processors running the simulation.

The *first* keyword allows a group to be specified whose atoms will be maintained as the first atoms in each processor's list of owned atoms. This in only useful when the specified group is a small fraction of all the atoms, and there are other operations LAMMPS is performing that will be sped-up significantly by being able to loop over the smaller set of atoms. Otherwise the reordering required by this option will be a net slow-down. The neigh_modify include and communicate group commands are two examples of commands that require this setting to work efficiently. Several fixes, most notably time integration fixes like fix nve, also take advantage of this setting if the group they operate on is the group specified by this command. Note that specifying "all" as the group-ID effectively turns off the *first* option.

It is OK to use the *first* keyword with a group that has not yet been defined, e.g. to use the atom_modify first command at the beginning of your input script. LAMMPS does not use the group until a simullation is run.

The *sort* keyword turns on a spatial sorting or reordering of atoms within each processor's sub-domain every *Nfreq* timesteps. If *Nfreq* is set to 0, then sorting is turned off. Sorting can improve cache performance and thus speed-up a LAMMPS simulation, as discussed in a paper by (Meloni). Its efficacy depends on the problem size (atoms/processor), how quickly the system becomes disordered, and various other factors. As a general rule, sorting is typically more effective at speeding up simulations of liquids as opposed to solids. In tests we have done, the speed-up can range from zero to 3-4x.

Reordering is peformed every *Nfreq* timesteps during a dynamics run or iterations during a minimization. More precisely, reordering occurs at the first reneighboring that occurs after the target timestep. The reordering is performed locally by each processor, using bins of the specified *binsize*. If *binsize* is set to 0.0, then a binsize equal to half the neighbor cutoff distance (force cutoff plus skin distance) is used, which is a reasonable value. After the atoms have been binned, they are reordered so that atoms in the same bin are adjacent to each other in the processor's 1d list of atoms.

The goal of this procedure is for atoms to put atoms close to each other in the processor's one−dimensional list of atoms that are also near to each other spatially. This can improve cache performance when pairwise intereractions and neighbor lists are computed. Note that if bins are too small, there will be few atoms/bin. Likewise if bins are too large, there will be many atoms/bin. In both cases, the goal of cache locality will be undermined.

IMPORTANT NOTE: Running a simulation with sorting on versus off should not change the simulation results in a statistical sense. However, a different ordering will induce round−off differences, which will lead to diverging trajectories over time when comparing two simluations. Various commands, particularly those which use random numbers (e.g. velocity create, and fix langevin), may generate (statistically identical) results which depend on the order in which atoms are processed. The order of atoms in a dump file will also typically change if sorting is enabled.

Restrictions:

The map keyword can only be used before the simulation box is defined by a read_data or create_box command.

The *first* and *sort* options cannot be used together. Since sorting is on by default, it will be turned off if the *first* keyword is used with a group−ID that is not "all".

Related commands: none

Default:

By default, atomic (non−molecular) problems do not allocate maps. For molecular problems, the option default is map = array. By default, a "first" group is not defined. By default, sorting is enabled with a frequency of 1000 and a binsize of 0.0, which means the neighbor cutoff will be used to set the bin size.

(Meloni) Meloni and Rasati, J Chem Phys, 126, 121102 (2007).

atom_style command

Syntax:

```
atom_style style args
```

- style = *angle* or *atomic* or *bond* or *charge* or *dipole* or *electron* or *ellipsoid* or *full* or *molecular* or *peri* or *sphere* or *hybrid*

```
  args = none for any style except hybrid
  hybrid args = list of one or more sub-styles
```

Examples:

```
atom_style atomic
atom_style bond
atom_style full
atom_style hybrid charge bond
```

Description:

Define what style of atoms to use in a simulation. This determines what attributes are associated with the atoms. This command must be used before a simulation is setup via a read_data, read_restart, or create_box command.

Once a style is assigned, it cannot be changed, so use a style general enough to encompass all attributes. E.g. with style *bond*, angular terms cannot be used or added later to the model. It is OK to use a style more general than needed, though it may be slightly inefficient.

The choice of style affects what quantities are stored by each atom, what quantities are communicated between processors to enable forces to be computed, and what quantities are listed in the data file read by the read_data command.

These are the additional attributes of each style and the typical kinds of physical systems they are used to model. All styles store coordinates, velocities, atom IDs and types. See the read_data, create_atoms, and set commands for info on how to set these various quantities.

angle	bonds and angles	bead−spring polymers with stiffness
atomic	only the default values	coarse−grain liquids, solids, metals
bond	bonds	bead−spring polymers
charge	charge	atomic system with charges
dipole	charge and dipole moment	system with dipolar particles
electron	charge and spin and eradius	electronic force field
ellipsoid	shape, quaternion for particle orientation, angular momentum	extended aspherical particles
full	molecular + charge	bio−molecules
molecular	bonds, angles, dihedrals, impropers	uncharged molecules
peri	mass, volume	mesocopic Peridynamic models
sphere	diameter, mass, angular velocity	granular models
wavepacket	charge, spin, eradius, etag, cs_re, cs_im	AWPMD

All of the styles assign mass to particles on a per−type basis, using the mass command, except for the finite−size particle styles discussed below. They assign mass on a per−atom basis.

All of the styles define point particles, except the *sphere*, *ellipsoid*, *electron*, *peri*, and *wavepacket* styles, which define finite−size particles.

For the *sphere* style, the particles are spheres and each stores a per−particle diameter and mass. If the diameter > 0.0, the particle is a finite−size sphere. If the diameter = 0.0, it is a point particle.

For the *ellipsoid* style, the particles are ellipsoids and each stores a flag which indicates whether it is a finite−size ellipsoid or a point particle. If it is an ellipsoid, it also stores a shape vector with the 3 diamters of the ellipsoid and a quaternion 4−vector with its orientation.

For the *electron* style, the particles representing electrons are 3d Gaussians with a specified position and bandwidth or uncertainty in position, which is represented by the eradius = electron size.

For the *peri* style, the particles are spherical and each stores a per−particle mass and volume.

The *wavepacket* style is similar to *electron*, but the electrons may consist of several Gaussian wave packets, summed up with coefficients cs= (cs_re,cs_im). Each of the wave packets is treated as a separate particle in LAMMPS, wave packets belonging to the same electron must have identical *etag* values.

Typically, simulations require only a single (non−hybrid) atom style. If some atoms in the simulation do not have all the properties defined by a particular style, use the simplest style that defines all the needed properties by any atom. For example, if some atoms in a simulation are charged, but others are not, use the *charge* style. If some atoms have bonds, but others do not, use the *bond* style.

The only scenario where the *hybrid* style is needed is if there is no single style which defines all needed properties of all atoms. For example, if you want dipolar particles which will be torqued and rotate, you would need to use "atom_style hybrid sphere dipole". When a hybrid style is used, atoms store and communicate the union of all quantities implied by the individual styles.

LAMMPS can be extended with new atom styles; see this section.

Restrictions:

This command cannot be used after the simulation box is defined by a read_data or create_box command.

The *angle*, *bond*, *full*, and *molecular* styles are part of the "molecular" package. The *dipole* style is part of the "dipole" package. The *ellipsoid* style is part of the "asphere" package. The *peri* style is part of the "peri" package for Peridynamics. The *electron* style is part of the "user−eff" package for electronic force fields. The *wavepacket* style is part of the "user−awpmd" package for the antisymmetrized wave packet MD method. They are only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

Related commands:

read_data, pair_style

Default:

atom_style atomic

bond_style class2 command

Syntax:

```
bond_style class2
```

Examples:

```
bond_style class2
bond_coeff 1 1.0 100.0 80.0 80.0
```

Description:

The *class2* bond style uses the potential

$$E = K_2(r - r_0)^2 + K_3(r - r_0)^3 + K_4(r - r_0)^4$$

where r0 is the equilibrium bond distance.

See (Sun) for a description of the COMPASS class2 force field.

The following coefficients must be defined for each bond type via the bond_coeff command as in the example above, or in the data file or restart files read by the read_data or read_restart commands:

- R0 (distance)
- K2 (energy/distance^2)
- K3 (energy/distance^3)
- K4 (energy/distance^4)

Restrictions:

This bond style can only be used if LAMMPS was built with the "class2" package. See the Making LAMMPS section for more info on packages.

Related commands:

bond_coeff, delete_bonds

Default: none

(Sun) Sun, J Phys Chem B 102, 7338−7364 (1998).

bond_coeff command

Syntax:

```
bond_coeff N args
```

- N = bond type (see asterisk form below)
- args = coefficients for one or more bond types

Examples:

```
bond_coeff 5 80.0 1.2
bond_coeff * 30.0 1.5 1.0 1.0
bond_coeff 1*4 30.0 1.5 1.0 1.0
bond_coeff 1 harmonic 200.0 1.0
```

Description:

Specify the bond force field coefficients for one or more bond types. The number and meaning of the coefficients depends on the bond style. Bond coefficients can also be set in the data file read by the read_data command or in a restart file.

N can be specified in one of two ways. An explicit numeric value can be used, as in the 1st example above. Or a wild−card asterisk can be used to set the coefficients for multiple bond types. This takes the form "*" or "*n" or "n*" or "m*n". If N = the number of bond types, then an asterisk with no numeric values means all types from 1 to N. A leading asterisk means all types from 1 to n (inclusive). A trailing asterisk means all types from n to N (inclusive). A middle asterisk means all types from m to n (inclusive).

Note that using a bond_coeff command can override a previous setting for the same bond type. For example, these commands set the coeffs for all bond types, then overwrite the coeffs for just bond type 2:

```
bond_coeff * 100.0 1.2
bond_coeff 2 200.0 1.2
```

A line in a data file that specifies bond coefficients uses the exact same format as the arguments of the bond_coeff command in an input script, except that wild−card asterisks should not be used since coefficients for all N types must be listed in the file. For example, under the "Bond Coeffs" section of a data file, the line that corresponds to the 1st example above would be listed as

```
5 80.0 1.2
```

Here is an alphabetic list of bond styles defined in LAMMPS. Click on the style to display the formula it computes and coefficients specified by the associated bond_coeff command:

- bond_style none − turn off bonded interactions
- bond_style hybrid − define multiple styles of bond interactions

- bond_style class2 − COMPASS (class 2) bond
- bond_style fene − FENE (finite−extensible non−linear elastic) bond
- bond_style fene/expand − FENE bonds with variable size particles
- bond_style harmonic − harmonic bond

- bond_style morse − Morse bond
- bond_style nonlinear − nonlinear bond
- bond_style quartic − breakable quartic bond
- bond_style table − tabulated by bond length

There are also additional bond styles submitted by users which are included in the LAMMPS distribution. The list of these with links to the individual styles are given in the bond section of this page.

Restrictions:

This command must come after the simulation box is defined by a read_data, read_restart, or create_box command.

A bond style must be defined before any bond coefficients are set, either in the input script or in a data file.

Related commands:

bond_style

Default: none

bond_style fene command

Syntax:

```
bond_style fene
```

Examples:

```
bond_style fene
bond_coeff 1 30.0 1.5 1.0 1.0
```

Description:

The *fene* bond style uses the potential

$$E = -0.5 K R_0^2 \ln \left[1 - \left(\frac{r}{R_0} \right)^2 \right] + 4\epsilon \left[\left(\frac{\sigma}{r} \right)^{12} - \left(\frac{\sigma}{r} \right)^6 \right] + \epsilon$$

to define a finite extensible nonlinear elastic (FENE) potential (Kremer), used for bead−spring polymer models. The first term is attractive, the 2nd Lennard−Jones term is repulsive. The first term extends to R0, the maximum extent of the bond. The 2nd term is cutoff at 2^(1/6) sigma, the minimum of the LJ potential.

The following coefficients must be defined for each bond type via the bond_coeff command as in the example above, or in the data file or restart files read by the read_data or read_restart commands:

- K (energy/distance^2)
- R0 (distance)
- epsilon (energy)
- sigma (distance)

Restrictions:

This bond style can only be used if LAMMPS was built with the "molecular" package (which it is by default). See the Making LAMMPS section for more info on packages.

You typically should specify special_bonds fene or special_bonds lj/coul 0 1 1 to use this bond style. LAMMPS will issue a warning it that's not the case.

Related commands:

bond_coeff, delete_bonds

Default: none

(Kremer) Kremer, Grest, J Chem Phys, 92, 5057 (1990).

bond_style fene/expand command

Syntax:

```
bond_style fene/expand
```

Examples:

```
bond_style fene/expand
bond_coeff 1 30.0 1.5 1.0 1.0 0.5
```

Description:

The *fene/expand* bond style uses the potential

$$E = -0.5 K R_0^2 \ln \left[1 - \left(\frac{(r-\Delta)}{R_0}\right)^2\right] + 4\epsilon \left[\left(\frac{\sigma}{(r-\Delta)}\right)^{12} - \left(\frac{\sigma}{(r-\Delta)}\right)^{6}\right] + \epsilon$$

to define a finite extensible nonlinear elastic (FENE) potential (Kremer), used for bead−spring polymer models. The first term is attractive, the 2nd Lennard−Jones term is repulsive.

The *fene/expand* bond style is similar to *fene* except that an extra shift factor of delta (positive or negative) is added to *r* to effectively change the bead size of the bonded atoms. The first term now extends to R0 + delta and the 2nd term is cutoff at 2^(1/6) sigma + delta.

The following coefficients must be defined for each bond type via the bond_coeff command as in the example above, or in the data file or restart files read by the read_data or read_restart commands:

- K (energy/distance^2)
- R0 (distance)
- epsilon (energy)
- sigma (distance)
- delta (distance)

Restrictions:

This bond style can only be used if LAMMPS was built with the "molecular" package (which it is by default). See the Making LAMMPS section for more info on packages.

You typically should specify special_bonds fene or special_bonds lj/coul 0 1 1 to use this bond style. LAMMPS will issue a warning it that's not the case.

Related commands:

bond_coeff, delete_bonds

Default: none

(Kremer) Kremer, Grest, J Chem Phys, 92, 5057 (1990).

bond_style harmonic command

Syntax:

```
bond_style harmonic
```

Examples:

```
bond_style harmonic
bond_coeff 5 80.0 1.2
```

Description:

The *harmonic* bond style uses the potential

$$E = K(r - r_0)^2$$

where r0 is the equilibrium bond distance. Note that the usual 1/2 factor is included in K.

The following coefficients must be defined for each bond type via the bond_coeff command as in the example above, or in the data file or restart files read by the read_data or read_restart commands:

- K (energy/distance^2)
- r0 (distance)

Restrictions:

This bond style can only be used if LAMMPS was built with the "molecular" package (which it is by default). See the Making LAMMPS section for more info on packages.

Related commands:

bond_coeff, delete_bonds

Default: none

bond_style harmonic/shift command

Syntax:

```
bond_style harmonic/shift
```

Examples:

```
bond_style harmonic/shift
bond_coeff 5 10.0 0.5 1.0
```

Description:

The *harmonic/shift* bond style is a shifted harmonic bond that uses the potential

$$E = \frac{Umin}{(r_0 - r_c)^2} \left[(r - r_0)^2 - (r_c - r_0)^2 \right]$$

where r0 is the equilibrium bond distance, and rc the critical distance. The potential is −Umin at r0 and zero at rc. The spring constant is k = Umin / [2 (r0−rc)^2].

The following coefficients must be defined for each bond type via the bond_coeff command as in the example above, or in the data file or restart files read by the read_data or read_restart commands:

- Umin (energy)
- r0 (distance)
- rc (distance)

Restrictions:

This bond style can only be used if LAMMPS was built with the "user−misc" package. See the Making LAMMPS section for more info on packages.

Related commands:

bond_coeff, delete_bonds, bond_harmonic

Default: none

bond_style harmonic/shift/cut command

Syntax:

```
bond_style harmonic/shift/cut
```

Examples:

```
bond_style harmonic/shift/cut
bond_coeff 5 10.0 0.5 1.0
```

Description:

The *harmonic/shift/cut* bond style is a shifted harmonic bond that uses the potential

$$E = \frac{Umin}{(r_0 - r_c)^2} \left[(r - r_0)^2 - (r_c - r_0)^2 \right]$$

where r0 is the equilibrium bond distance, and rc the critical distance. The bond potential is zero for distances r > rc. The potential is −Umin at r0 and zero at rc. The spring constant is k = Umin / [2 (r0−rc)^2].

The following coefficients must be defined for each bond type via the bond_coeff command as in the example above, or in the data file or restart files read by the read_data or read_restart commands:

- Umin (energy)
- r0 (distance)
- rc (distance)

Restrictions:

This bond style can only be used if LAMMPS was built with the "user−misc" package. See the Making LAMMPS section for more info on packages.

Related commands:

bond_coeff, delete_bonds, bond_harmonic, bond_harmonicshift

Default: none

bond_style hybrid command

Syntax:

```
bond_style hybrid style1 style2 ...
```

- style1,style2 = list of one or more bond styles

Examples:

```
bond_style hybrid harmonic fene
bond_coeff 1 harmonic 80.0 1.2
bond_coeff 2* fene 30.0 1.5 1.0 1.0
```

Description:

The *hybrid* style enables the use of multiple bond styles in one simulation. A bond style is assigned to each bond type. For example, bonds in a polymer flow (of bond type 1) could be computed with a *fene* potential and bonds in the wall boundary (of bond type 2) could be computed with a *harmonic* potential. The assignment of bond type to style is made via the bond_coeff command or in the data file.

In the bond_coeff commands, the name of a bond style must be added after the bond type, with the remaining coefficients being those appropriate to that style. In the example above, the 2 bond_coeff commands set bonds of bond type 1 to be computed with a *harmonic* potential with coefficients 80.0, 1.2 for K, r0. All other bond types (2−N) are computed with a *fene* potential with coefficients 30.0, 1.5, 1.0, 1.0 for K, R0, epsilon, sigma.

If bond coefficients are specified in the data file read via the read_data command, then the same rule applies. E.g. "harmonic" or "fene" must be added after the bond type, for each line in the "Bond Coeffs" section, e.g.

```
Bond Coeffs

1 harmonic 80.0 1.2
2 fene 30.0 1.5 1.0 1.0
...
```

A bond style of *none* with no additional coefficients can be used in place of a bond style, either in a input script bond_coeff command or in the data file, if you desire to turn off interactions for specific bond types.

Restrictions:

This bond style can only be used if LAMMPS was built with the "molecular" package (which it is by default). See the Making LAMMPS section for more info on packages.

Unlike other bond styles, the hybrid bond style does not store bond coefficient info for individual sub−styles in a binary restart files. Thus when retarting a simulation from a restart file, you need to re−specify bond_coeff commands.

Related commands:

bond_coeff, delete_bonds

Default: none

bond_style morse command

Syntax:

```
bond_style morse
```

Examples:

```
bond_style morse
bond_coeff 5 1.0 2.0 1.2
```

Description:

The *morse* bond style uses the potential

$$E = D\left[1 - e^{-\alpha(r-r0)}\right]^2$$

where r0 is the equilibrium bond distance, alpha is a stiffness parameter, and D determines the depth of the potential well.

The following coefficients must be defined for each bond type via the bond_coeff command as in the example above, or in the data file or restart files read by the read_data or read_restart commands:

- D (energy)
- alpha (inverse distance)
- r0 (distance)

Restrictions:

This bond style can only be used if LAMMPS was built with the "molecular" package (which it is by default). See the Making LAMMPS section for more info on packages.

Related commands:

bond_coeff, delete_bonds

Default: none

bond_style none command

Syntax:

```
bond_style none
```

Examples:

```
bond_style none
```

Description:

Using a bond style of none means bond forces are not computed, even if pairs of bonded atoms were listed in the data file read by the read_data command.

Restrictions: none

Related commands: none

Default: none

bond_style nonlinear command

Syntax:

```
bond_style nonlinear
```

Examples:

```
bond_style nonlinear
bond_coeff 2 100.0 1.1 1.4
```

Description:

The *nonlinear* bond style uses the potential

$$E = \frac{\epsilon(r - r_0)^2}{[\lambda^2 - (r - r_0)^2]}$$

to define an anharmonic spring (Rector) of equilibrium length r0 and maximum extension lamda.

The following coefficients must be defined for each bond type via the bond_coeff command as in the example above, or in the data file or restart files read by the read_data or read_restart commands:

- epsilon (energy)
- r0 (distance)
- lamda (distance)

Restrictions:

This bond style can only be used if LAMMPS was built with the "molecular" package (which it is by default). See the Making LAMMPS section for more info on packages.

Related commands:

bond_coeff, delete_bonds

Default: none

(Rector) Rector, Van Swol, Henderson, Molecular Physics, 82, 1009 (1994).

bond_style quartic command

Syntax:

```
bond_style quartic
```

Examples:

```
bond_style quartic
bond_coeff 2 1200 −0.55 0.25 1.3 34.6878
```

Description:

The *quartic* bond style uses the potential

$$E = K(r - R_c)^2(r - R_c - B_1)(r - R_c - B_2) + U_0 + 4\epsilon \left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right] + \epsilon$$

to define a bond that can be broken as the simulation proceeds (e.g. due to a polymer being stretched). The sigma and epsilon used in the LJ portion of the formula are both set equal to 1.0 by LAMMPS.

The following coefficients must be defined for each bond type via the bond_coeff command as in the example above, or in the data file or restart files read by the read_data or read_restart commands:

- K (energy/distance^2)
- B1 (distance)
- B2 (distance)
- Rc (distance)
- U0 (energy)

This potential was constructed to mimic the FENE bond potential for coarse−grained polymer chains. When monomers with sigma = epsilon = 1.0 are used, the following choice of parameters gives a quartic potential that looks nearly like the FENE potential: K = 1200, B1 = −0.55, B2 = 0.25, Rc = 1.3, and U0 = 34.6878. Different parameters can be specified using the bond_coeff command, but you will need to choose them carefully so they form a suitable bond potential.

Rc is the cutoff length at which the bond potential goes smoothly to a local maximum. If a bond length ever becomes > Rc, LAMMPS "breaks" the bond, which means two things. First, the bond potential is turned off by setting its type to 0, and is no longer computed. Second, a pairwise interaction between the two atoms is turned on, since they are no longer bonded.

LAMMPS does the second task via a computational sleight−of−hand. It subtracts the pairwise interaction as part of the bond computation. When the bond breaks, the subtraction stops. For this to work, the pairwise interaction must always be computed by the pair_style command, whether the bond is broken or not. This means that special_bonds must be set to 1,1,1, as indicated as a restriction below.

Note that when bonds are dumped to a file via the dump local command, bonds with type 0 are not included. The delete_bonds command can also be used to query the status of broken bonds or permanently delete them, e.g.:

```
delete_bonds all stats
```

```
delete_bonds all bond 0 remove
```

Restrictions:

This bond style can only be used if LAMMPS was built with the "molecular" package (which it is by default). See the Making LAMMPS section for more info on packages.

The *quartic* style requires that special_bonds parameters be set to 1,1,1. Three− and four−body interactions (angle, dihedral, etc) cannot be used with *quartic* bonds.

Related commands:

bond_coeff, delete_bonds

Default: none

bond_style command

Syntax:

```
bond_style style args
```

- style = *none* or *hybrid* or *class2* or *fene* or *fene/expand* or *harmonic* or *morse* or *nonlinear* or *quartic*

```
args = none for any style except hybrid
hybrid args = list of one or more styles
```

Examples:

```
bond_style harmonic
bond_style fene
bond_style hybrid harmonic fene
```

Description:

Set the formula(s) LAMMPS uses to compute bond interactions between pairs of atoms. In LAMMPS, a bond differs from a pairwise interaction, which are set via the pair_style command. Bonds are defined between specified pairs of atoms and remain in force for the duration of the simulation (unless the bond breaks which is possible in some bond potentials). The list of bonded atoms is read in by a read_data or read_restart command from a data or restart file. By contrast, pair potentials are typically defined between all pairs of atoms within a cutoff distance and the set of active interactions changes over time.

Hybrid models where bonds are computed using different bond potentials can be setup using the *hybrid* bond style.

The coefficients associated with a bond style can be specified in a data or restart file or via the bond_coeff command.

All bond potentials store their coefficient data in binary restart files which means bond_style and bond_coeff commands do not need to be re−specified in an input script that restarts a simulation. See the read_restart command for details on how to do this. The one exception is that bond_style *hybrid* only stores the list of sub−styles in the restart file; bond coefficients need to be re−specified.

IMPORTANT NOTE: When both a bond and pair style is defined, the special_bonds command often needs to be used to turn off (or weight) the pairwise interaction that would otherwise exist between 2 bonded atoms.

In the formulas listed for each bond style, *r* is the distance between the 2 atoms in the bond.

Here is an alphabetic list of bond styles defined in LAMMPS. Click on the style to display the formula it computes and coefficients specified by the associated bond_coeff command:

- bond_style none − turn off bonded interactions
- bond_style hybrid − define multiple styles of bond interactions

- bond_style class2 − COMPASS (class 2) bond
- bond_style fene − FENE (finite−extensible non−linear elastic) bond
- bond_style fene/expand − FENE bonds with variable size particles

- bond_style harmonic − harmonic bond
- bond_style morse − Morse bond
- bond_style nonlinear − nonlinear bond
- bond_style quartic − breakable quartic bond
- bond_style table − tabulated by bond length

There are also additional bond styles submitted by users which are included in the LAMMPS distribution. The list of these with links to the individual styles are given in the bond section of this page.

Restrictions:

Bond styles can only be set for atom styles that allow bonds to be defined.

Most bond styles are part of the "molecular" package. They are only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info on packages. The doc pages for individual bond potentials tell if it is part of a package.

Related commands:

bond_coeff, delete_bonds

Default:

bond_style none

bond_style table command

Syntax:

```
bond_style table style N
```

- style = *linear* or *spline* = method of interpolation
- N = use N values in table

Examples:

```
bond_style table linear 1000
bond_coeff 1 file.table ENTRY1
```

Description:

Style *table* creates interpolation tables of length *N* from bond potential and force values listed in a file(s) as a function of bond length. The files are read by the bond_coeff command.

The interpolation tables are created by fitting cubic splines to the file values and interpolating energy and force values at each of *N* distances. During a simulation, these tables are used to interpolate energy and force values as needed. The interpolation is done in one of 2 styles: *linear* or *spline*.

For the *linear* style, the bond length is used to find 2 surrounding table values from which an energy or force is computed by linear interpolation.

For the *spline* style, a cubic spline coefficients are computed and stored at each of the *N* values in the table. The bond length is used to find the appropriate set of coefficients which are used to evaluate a cubic polynomial which computes the energy or force.

The following coefficients must be defined for each bond type via the bond_coeff command as in the example above.

- filename
- keyword

The filename specifies a file containing tabulated energy and force values. The keyword specifies a section of the file. The format of this file is described below.

The format of a tabulated file is as follows (without the parenthesized comments):

```
# Bond potential for harmonic (one or more comment or blank lines)

HAM                           (keyword is the first text on line)
N 101 FP 0 0 EQ 0.5           (N, FP, EQ  parameters)
                              (blank line)
1 0.00 338.0000 1352.0000     (index, bond-length, energy, force)
2 0.01 324.6152 1324.9600
...
101 1.00 338.0000 -1352.0000
```

A section begins with a non−blank line whose 1st character is not a "#"; blank lines or lines starting with "#" can be used as comments between sections. The first line begins with a keyword which identifies the section. The line can contain additional text, but the initial text must match the argument specified in the bond_coeff command. The next line lists (in any order) one or more parameters for the table. Each parameter is a keyword followed by one or more numeric values.

The parameter "N" is required and its value is the number of table entries that follow. Note that this may be different than the *N* specified in the bond_style table command. Let Ntable = *N* in the bond_style command, and Nfile = "N" in the tabulated file. What LAMMPS does is a preliminary interpolation by creating splines using the Nfile tabulated values as nodal points. It uses these to interpolate as needed to generate energy and force values at Ntable different points. The resulting tables of length Ntable are then used as described above, when computing energy and force for individual bond lengths. This means that if you want the interpolation tables of length Ntable to match exactly what is in the tabulated file (with effectively no preliminary interpolation), you should set Ntable = Nfile.

The "FP" parameter is optional. If used, it is followed by two values fplo and fphi, which are the derivatives of the force at the innermost and outermost bond lengths. These values are needed by the spline construction routines. If not specified by the "FP" parameter, they are estimated (less accurately) by the first two and last two force values in the table.

The "EQ" parameter is also optional. If used, it is followed by a the equilibrium bond length, which is used, for example, by the fix shake command. If not used, the equilibrium bond length is set to 0.0.

Following a blank line, the next N lines list the tabulated values. On each line, the 1st value is the index from 1 to N, the 2nd value is the bond length r (in distance units), the 3rd value is the energy (in energy units), and the 4th is the force (in force units). The bond lengths must range from a LO value to a HI value, and increase from one line to the next. If the actual bond length is ever smaller than the LO value or larger than the HI value, then the bond energy and force is evaluated as if the bond were the LO or HI length.

Note that one file can contain many sections, each with a tabulated potential. LAMMPS reads the file section by section until it finds one that matches the specified keyword.

Restrictions:

This bond style can only be used if LAMMPS was built with the "molecular" package (which it is by default). See the Making LAMMPS section for more info on packages.

Related commands:

bond_coeff, delete_bonds

Default: none

boundary command

Syntax:

```
boundary x y z
```

- x,y,z = *p* or *s* or *f* or *m*, one or two letters

```
  p is periodic
    f is non-periodic and fixed
    s is non-periodic and shrink-wrapped
    m is non-periodic and shrink-wrapped with a minimum value
```

Examples:

```
boundary p p f
boundary p fs p
boundary s f fm
```

Description:

Set the style of boundaries for the global simulation box in each dimension. A single letter assigns the same style to both the lower and upper face of the box. Two letters assigns the first style to the lower face and the second style to the upper face. The initial size of the simulation box is set by the read_data, read_restart, or create_box commands.

The style *p* means the box is periodic, so that particles interact across the boundary, and they can exit one end of the box and re−enter the other end. A periodic dimension can change in size due to constant pressure boundary conditions or box deformation (see the fix npt and fix deform commands). The *p* style must be applied to both faces of a dimension.

The styles *f*, *s*, and *m* mean the box is non−periodic, so that particles do not interact across the boundary and do not move from one side of the box to the other. For style *f*, the position of the face is fixed. If an atom moves outside the face it may be lost. For style *s*, the position of the face is set so as to encompass the atoms in that dimension (shrink−wrapping), no matter how far they move. For style *m*, shrink−wrapping occurs, but is bounded by the value specified in the data or restart file or set by the create_box command. For example, if the upper z face has a value of 50.0 in the data file, the face will always be positioned at 50.0 or above, even if the maximum z−extent of all the atoms becomes less than 50.0.

Restrictions:

This command cannot be used after the simulation box is defined by a read_data or create_box command.

Related commands:

See the thermo_modify command for a discussion of lost atoms.

Default:

```
boundary p p p
```

change_box command

Syntax:

```
change_box style
```

- style = *ortho* or *triclinic*

```
  ortho = convert simulation box from non-orthogonal (triclinic) to orthogonal
    triclinic = convert simulation box from orthogonal to non-orthogonal (triclinic)
```

Examples:

```
change_box ortho
change_box triclinic
```

Description:

By default LAMMPS runs a simulation in an orthogonal, axis−aligned simulation box. LAMMPS can also run simulations in non−orthogonal (triclinic) simulation boxes. A box is defined as either orthogonal or non−orthogonal when it is created via the create_box, read_data, or read_restart commands.

This command allows you to toggle the existing simulation box from orthogonal to non−orthogonal and vice versa. For example, an initial equilibration simulation can be run in an orthogonal box, the box can be toggled to non−orthogonal, and then a non−equilibrium MD (NEMD) simulation can be run with deformation via the fix deform command.

Note that if the simulation box is currently non−orthogonal and has non−zero tilt in xy, yz, or xz, then it cannot be converted to an orthogonal box.

Restrictions:

At the point in the input script when this command is issued, no dumps can be active, nor can a fix ave/spatial or fix deform be active. This is because these commands test whether the simulation box is orthogonal when they are first issued. Note that these commands can appear in your script before a change_box command is issued, so long as an undump or unfix command is also used to turn them off.

Related commands: none

Default: none

clear command

Syntax:

```
clear
```

Examples:

```
(commands for 1st simulation)
clear
(commands for 2nd simulation)
```

Description:

This command deletes all atoms, restores all settings to their default values, and frees all memory allocated by LAMMPS. Once a clear command has been executed, it is as if LAMMPS were starting over, with only the exceptions noted below. This command enables multiple jobs to be run sequentially from one input script.

These settings are not affected by a clear command: the working directory (shell command), log file status (log command), echo status (echo command), and input script variables (variable command).

Restrictions: none

Related commands: none

Default: none

communicate command

Syntax:

```
communicate style keyword value ...
```

- style = *single* or *multi*
- zero or more keyword/value pairs may be appended
- keyword = *cutoff* or *group* or *vel*

```
  cutoff value = Rcut (distance units) = communicate atoms from this far away
    group value = group-ID = only communicate atoms in the group
    vel value = yes or no = do or do not communicate velocity info with ghost atoms
```

Examples:

```
communicate multi
communicate multi group solvent
communicate single vel yes
communicate single cutoff 5.0 vel yes
```

Description:

This command sets the style of inter−processor communication that occurs each timestep as atom coordinates and other properties are exchanged between neighboring processors and stored as properties of ghost atoms.

The default style is *single* which means each processor acquires information for ghost atoms that are within a single distance from its sub−domain. The distance is the maximum of the neighbor cutoff for all atom type pairs.

For many systems this is an efficient algorithm, but for systems with widely varying cutoffs for different type pairs, the *multi* style can be faster. In this case, each atom type is assigned its own distance cutoff for communication purposes, and fewer atoms will be communicated. See the neighbor multi command for a neighbor list construction option that may also be beneficial for simulations of this kind.

The *cutoff* option allows you to set a ghost cutoff distance, which is the distance from the borders of a processor's sub−domain at which ghost atoms are acquired from other processors. By default the ghost cutoff = neighbor cutoff = pairwise force cutoff + neighbor skin. See the neighbor command for more information about the skin distance. If the specified Rcut is greater than the neighbor cutoff, then extra ghost atoms will be acquired. If it is smaller, the ghost cutoff is set to the neighbor cutoff.

These are simulation scenarios in which it may be useful to set a ghost cutoff > neighbor cutoff:

- a single polymer chain with bond interactions, but no pairwise interactions
- bonded interactions (e.g. dihedrals) extend further than the pairwise cutoff
- ghost atoms beyond the pairwise cutoff are needed for some computation

In the first scenario, a pairwise potential may not be defined. Thus the pairwise neighbor cutoff will be 0.0. But ghost atoms are still needed for computing bond, angle, etc interactions between atoms on different processors. The appropriate ghost cutoff depends on the newton bond setting. For newton bond *off*, the distance needs to be the furthest distance between any two atoms in the bond, angle, etc. E.g. the distance between 1−4 atoms in a dihedral. For newton bond *on*, the distance between the central atom in the bond, angle, etc and any other atom is

sufficient. E.g. the distance between 2−4 atoms in a dihedral.

In the second scenario, a pairwise potential is defined, but its neighbor cutoff is not sufficiently long enough to enable bond, angle, etc terms to be computed. As in the previous scenario, an appropriate ghost cutoff should be set.

In the last scenario, a fix or compute or pairwise potential needs to calculate with ghost atoms beyond the normal pairwise cutoff for some computation it performs (e.g. locate neighbors of ghost atoms in a multibody pair potential). Setting the ghost cutoff appropriately can insure it will find the needed atoms.

The *group* option will limit communication to atoms in the specified group. This can be useful for models where no ghost atoms are needed for some kinds of particles. All atoms (not just those in the specified group) will still migrate to new processors as they move. The group specified with this option must also be specified via the atom_modify first command.

The *vel* option enables velocity information to be communicated with ghost particles. Depending on the atom_style, velocity info includes the translational velocity, angular velocity, and angular momentum of a particle. If the *vel* option is set to *yes*, then ghost atoms store these quantities; if *no* then they do not. The *yes* setting is needed by some pair styles which require the velocity state of both the I and J particles to compute a pairwise I,J interaction.

Note that if the fix deform command is being used with its "remap v" option enabled, then the velocities for ghost atoms (in the fix deform group) mirrored across a periodic boundary will also include components due to any velocity shift that occurs across that boundary (e.g. due to dilation or shear).

Restrictions: none

Related commands:

neighbor

Default:

The default settings are style = single, group = all, cutoff = 0.0, vel = no. The cutoff default of 0.0 means that ghost cutoff = neighbor cutoff = pairwise force cutoff + neighbor skin.

compute command

Syntax:

```
compute ID group-ID style args
```

- ID = user−assigned name for the computation
- group−ID = ID of the group of atoms to perform the computation on
- style = one of a list of possible style names (see below)
- args = arguments used by a particular style

Examples:

```
compute 1 all temp
compute newtemp flow temp/partial 1 1 0
compute 3 all ke/atom
```

Description:

Define a computation that will be performed on a group of atoms. Quantities calculated by a compute are instantaneous values, meaning they are calculated from information about atoms on the current timestep or iteration, though a compute may internally store some information about a previous state of the system. Defining a compute does not perform a computation. Instead computes are invoked by other LAMMPS commands as needed, e.g. to calculate a temperature needed for a thermostat fix or to generate thermodynamic or dump file output. See this howto section for a summary of various LAMMPS output options, many of which involve computes.

The ID of a compute can only contain alphanumeric characters and underscores.

Computes calculate one of three styles of quantities: global, per−atom, or local. A global quantity is one or more system−wide values, e.g. the temperature of the system. A per−atom quantity is one or more values per atom, e.g. the kinetic energy of each atom. Per−atom values are set to 0.0 for atoms not in the specified compute group. Local quantities are calculated by each processor based on the atoms it owns, but there may be zero or more per atom, e.g. a list of bond distances. Computes that produce per−atom quantities have the word "atom" in their style, e.g. *ke/atom*. Computes that produce local quantities have the word "local" in their style, e.g. *bond/local*. Styles with neither "atom" or "local" in their style produce global quantities.

Note that a single compute produces either global or per−atom or local quantities, but never more than one of these.

Global, per−atom, and local quantities each come in three kinds: a single scalar value, a vector of values, or a 2d array of values. The doc page for each compute describes the style and kind of values it produces, e.g. a per−atom vector. Some computes produce more than one kind of a single style, e.g. a global scalar and a global vector.

When a compute quantity is accessed, as in many of the output commands discussed below, it can be referenced via the following bracket notation, where ID is the ID of the compute:

c_ID	entire scalar, vector, or array
c_ID[I]	one element of vector, one column of array
c_ID[I][J]	one element of array

In other words, using one bracket reduces the dimension of the quantity once (vector −> scalar, array −> vector). Using two brackets reduces the dimension twice (array −> scalar). Thus a command that uses scalar compute values as input can also process elements of a vector or array.

Note that commands and variables which use compute quantities typically do not allow for all kinds, e.g. a command may require a vector of values, not a scalar. This means there is no ambiguity about referring to a compute quantity as c_ID even if it produces, for example, both a scalar and vector. The doc pages for various commands explain the details.

In LAMMPS, the values generated by a compute can be used in several ways:

- The results of computes that calculate a global temperature or pressure can be used by fixes that do thermostatting or barostatting or when atom velocities are created.
- Global values can be output via the thermo_style custom or fix ave/time command. Or the values can be referenced in a variable equal or variable atom command.
- Per−atom values can be output via the dump custom command or the fix ave/spatial command. Or they can be time−averaged via the fix ave/atom command or reduced by the compute reduce command. Or the per−atom values can be referenced in an atom−style variable.
- Local values can be reduced by the compute reduce command, or histogrammed by the fix ave/histo command, or output by the dump local command.

The results of computes that calculate global quantities can be either "intensive" or "extensive" values. Intensive means the value is independent of the number of atoms in the simulation, e.g. temperature. Extensive means the value scales with the number of atoms in the simulation, e.g. total rotational kinetic energy. Thermodynamic output will normalize extensive values by the number of atoms in the system, depending on the "thermo_modify norm" setting. It will not normalize intensive values. If a compute value is accessed in another way, e.g. by a variable, you may want to know whether it is an intensive or extensive value. See the doc page for individual computes for further info.

LAMMPS creates its own computes internally for thermodynamic output. Three computes are always created, named "thermo_temp", "thermo_press", and "thermo_pe", as if these commands had been invoked in the input script:

```
compute thermo_temp all temp
compute thermo_press all pressure thermo_temp
compute thermo_pe all pe
```

Additional computes for other quantities are created if the thermo style requires it. See the documentation for the thermo_style command.

Fixes that calculate temperature or pressure, i.e. for thermostatting or barostatting, may also create computes. These are discussed in the documentation for specific fix commands.

In all these cases, the default computes LAMMPS creates can be replaced by computes defined by the user in the input script, as described by the thermo_modify and fix modify commands.

Properties of either a default or user−defined compute can be modified via the compute_modify command.

Computes can be deleted with the uncompute command.

Code for new computes can be added to LAMMPS (see this section of the manual) and the results of their calculations accessed in the various ways described above.

Each compute style has its own doc page which describes its arguments and what it does. Here is an alphabetic list of compute styles available in LAMMPS:

- angle/local − theta and energy of each angle
- atom/molecule − sum per−atom properties for each molecule
- bond/local − distance and energy of each bond
- centro/atom − centro−symmetry parameter for each atom
- cluster/atom − cluster ID for each atom
- cna/atom − common neighbor analysis (CNA) for each atom
- com − center−of−mass of group of atoms
- com/molecule − center−of−mass for each molecule
- coord/atom − coordination number for each atom
- damage/atom − Peridynamic damage for each atom
- dihedral/local − angle of each dihedral
- displace/atom − displacement of each atom
- erotate/asphere − rotational energy of aspherical particles
- erotate/sphere − rotational energy of spherical particles
- event/displace − detect event on atom displacement
- group/group − energy/force between two groups of atoms
- gyration − radius of gyration of group of atoms
- gyration/molecule − radius of gyration for each molecule
- heat/flux − heat flux through a group of atoms
- improper/local − angle of each improper
- ke − translational kinetic energy
- ke/atom − kinetic energy for each atom
- msd − mean−squared displacement of group of atoms
- msd/molecule − mean−squared displacement for each molecule
- pair − values computed by a pair style
- pair/local − distance/energy/force of each pairwise interaction
- pe − potential energy
- pe/atom − potential energy for each atom
- pressure − total pressure and pressure tensor
- property/atom − convert atom attributes to per−atom vectors/arrays
- property/local − convert local attributes to localvectors/arrays
- property/molecule − convert molecule attributes to localvectors/arrays
- rdf − radial distribution function g(r) histogram of group of atoms
- reduce − combine per−atom quantities into a single global value
- reduce/region − same as compute reduce, within a region
- slice − extract values from global vector or array
- stress/atom − stress tensor for each atom
- temp − temperature of group of atoms
- temp/asphere − temperature of aspherical particles
- temp/com − temperature after subtracting center−of−mass velocity
- temp/deform − temperature excluding box deformation velocity
- temp/partial − temperature excluding one or more dimensions of velocity
- temp/profile − temperature excluding a binned velocity profile
- temp/ramp − temperature excluding ramped velocity component
- temp/region − temperature of a region of atoms
- temp/sphere − temperature of spherical particles
- ti − thermodyanmic integration free energy values

There are also additional compute styles submitted by users which are included in the LAMMPS distribution. The list of these with links to the individual styles are given in the compute section of this page.

There are also additional accelerated compute styles included in the LAMMPS distribution for faster performance on CPUs and GPUs. The list of these with links to the individual styles are given in the pair section of this page.

Restrictions: none

Related commands:

uncompute, compute_modify, fix ave/atom, fix ave/spatial, fix ave/time, fix ave/histo

Default: none

compute ackland/atom command

Syntax:

```
compute ID group−ID ackland/atom
```

- ID, group−ID are documented in compute command
- ackland/atom = style name of this compute command

Examples:

```
compute 1 all ackland/atom
```

Description:

Defines a computation that calculates the local lattice structure according to the formulation given in (Ackland).

In contrast to the centro−symmetry parameter this method is stable against temperature boost, because it is based not on the distance between particles but the angles. Therefore statistical fluctuations are averaged out a little more. A comparison with the Common Neighbor Analysis metric is made in the paper.

The result is a number which is mapped to the following different lattice structures:

- 0 = UNKNOWN
- 1 = BCC
- 2 = FCC
- 3 = HCP
- 4 = ICO

The neighbor list needed to compute this quantity is constructed each time the calculation is performed (i.e. each time a snapshot of atoms is dumped). Thus it can be inefficient to compute/dump this quantity too frequently or to have multiple compute/dump commands, each of which computes this quantity.−

Output info:

This compute calculates a scalar quantity for each atom, which can be accessed by any command that uses per−atom values from a compute as input. See this section for an overview of LAMMPS output options.

Restrictions:

This compute is part of the "user−misc" package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

Related commands:

compute centro/atom

Default: none

(Ackland) Ackland, Jones, Phys Rev B, 73, 054104 (2006).

compute angle/local command

Syntax:

```
compute ID group-ID angle/local input1 input2 ...
```

- ID, group−ID are documented in compute command
- angle/local = style name of this compute command
- zero or more keywords may be appended
- keyword = *theta* or *eng*

```
  theta = tabulate angles
    eng = tabulate angle energies
```

Examples:

```
compute 1 all angle/local theta
compute 1 all angle/local eng theta
```

Description:

Define a computation that calculates properties of individual angle interactions. The number of datums generated, aggregated across all processors, equals the number of angles in the system.

The local data stored by this command is generated by looping over all the atoms owned on a processor and their angles. An angle will only be included if all 3 atoms in the angle are in the specified compute group. Any angles that have been broken (see the angle_style command) by setting their angle type to 0 are not included. Angles that have been turned off (see the fix shake or delete_bonds commands) by setting their angle type negative are written into the file, but their energy will be 0.0.

Note that as atoms migrate from processor to processor, there will be no consistent ordering of the entries within the local vector or array from one timestep to the next. The only consistency that is guaranteed is that the ordering on a particular timestep will be the same for local vectors or arrays generated by other compute commands. For example, angle output from the compute property/local command can be combined with data from this command and output by the dump local command in a consistent way.

Output info:

This compute calculates a local vector or local array depending on the number of keywords. The length of the vector or number of rows in the array is the number of angles. If a single keyword is specified, a local vector is produced. If two or more keywords are specified, a local array is produced where the number of columns = the number of keywords. The vector or array can be accessed by any command that uses local values from a compute as input. See this section for an overview of LAMMPS output options.

The output for *theta* will be in degrees. The output for *eng* will be in energy units.

Restrictions: none

Related commands:

dump local, compute property/local

Default: none

compute atom/molecule command

Syntax:

```
compute ID group-ID atom/molecule input1 input2 ...
```

- ID, group−ID are documented in compute command
- atom/molecule = style name of this compute command
- one or more inputs can be listed
- input = c_ID, c_ID[N], f_ID, f_ID[N], v_name

```
c_ID = per-atom vector calculated by a compute with ID
c_ID[I] = Ith column of per-atom array calculated by a compute with ID
f_ID = per-atom vector calculated by a fix with ID
f_ID[I] = Ith column of per-atom array calculated by a fix with ID
v_name = per-atom vector calculated by an atom-style variable with name
```

Examples:

```
compute 1 all atom/molecule c_ke c_pe
compute 1 top atom/molecule v_myFormula c_stress3
```

Description:

Define a calculation that sums per−atom values on a per−molecule basis, one per listed input. The inputs can computes, fixes, or variables that generate per−atom quantities. Note that attributes stored by atoms, such as mass or force, can also be summed on a per−molecule basis, by accessing these quantities via the compute property/atom command.

Each listed input is operated on independently. Only atoms within the specified group contribute to the per−molecule sum. Note that compute or fix inputs define their own group which may affect the quantities they return. For example, if a compute is used as an input which generates a per−atom vector, it will generate values of 0.0 for atoms that are not in the group specified for that compute.

The ordering of per−molecule quantities produced by this compute is consistent with the ordering produced by other compute commands that generate per−molecule datums. Conceptually, them molecule IDs will be in ascending order for any molecule with one or more of its atoms in the specified group.

If an input begins with "c_", a compute ID must follow which has been previously defined in the input script and which generates per−atom quantities. See the individual compute doc page for details. If no bracketed integer is appended, the vector calculated by the compute is used. If a bracketed interger is appended, the Ith column of the array calculated by the compute is used. Users can also write code for their own compute styles and add them to LAMMPS.

If an input begins with "f_", a fix ID must follow which has been previously defined in the input script and which generates per−atom quantities. See the individual fix doc page for details. Note that some fixes only produce their values on certain timesteps, which must be compatible with when compute atom/molecule references the values, else an error results. If no bracketed integer is appended, the vector calculated by the fix is used. If a bracketed integer is appended, the Ith column of the array calculated by the fix is used. Users can also write code for their own fix style and add them to LAMMPS.

If an input begins with "v_", a variable name must follow which has been previously defined in the input script. It must be an atom−style variable. Atom−style variables can reference thermodynamic keywords and various per−atom attributes, or invoke other computes, fixes, or variables when they are evaluated, so this is a very general means of generating per−atom quantities to sum on a per−molecule basis.

Output info:

This compute calculates a global vector or global array depending on the number of input values. The length of the vector or number of rows in the array is the number of molecules. If a single input is specified, a global vector is produced. If two or more inputs are specified, a global array is produced where the number of columns = the number of inputs. The vector or array can be accessed by any command that uses global values from a compute as input. See this section for an overview of LAMMPS output options.

All the vector or array values calculated by this compute are "extensive".

The vector or array values will be in whatever units the input quantities are in.

Restrictions: none

Related commands:

compute, fix, variable

Default: none

compute bond/local command

Syntax:

```
compute ID group-ID bond/local input1 input2 ...
```

- ID, group−ID are documented in compute command
- bond/local = style name of this compute command
- zero or more keywords may be appended
- keyword = *dist* or *eng*

```
  dist = tabulate bond distances
    eng = tablutate bond energies
```

Examples:

```
compute 1 all bond/local eng
compute 1 all bond/local dist eng
```

Description:

Define a computation that calculates properties of individual bond interactions. The number of datums generated, aggregated across all processors, equals the number of bonds in the system.

The local data stored by this command is generated by looping over all the atoms owned on a processor and their bonds. A bond will only be included if both atoms in the bond are in the specified compute group. Any bonds that have been broken (see the bond_style command) by setting their bond type to 0 are not included. Bonds that have been turned off (see the fix shake or delete_bonds commands) by setting their bond type negative are written into the file, but their energy will be 0.0.

Note that as atoms migrate from processor to processor, there will be no consistent ordering of the entries within the local vector or array from one timestep to the next. The only consistency that is guaranteed is that the ordering on a particular timestep will be the same for local vectors or arrays generated by other compute commands. For example, bond output from the compute property/local command can be combined with data from this command and output by the dump local command in a consistent way.

Output info:

This compute calculates a local vector or local array depending on the number of keywords. The length of the vector or number of rows in the array is the number of bonds. If a single keyword is specified, a local vector is produced. If two or more keywords are specified, a local array is produced where the number of columns = the number of keywords. The vector or array can be accessed by any command that uses local values from a compute as input. See this section for an overview of LAMMPS output options.

The output for *dist* will be in distance units. The output for *eng* will be in energy units.

Restrictions: none

Related commands:

dump local, compute property/local

Default: none

compute centro/atom command

Syntax:

```
compute ID group-ID centro/atom lattice
```

- ID, group−ID are documented in compute command
- centro/atom = style name of this compute command
- lattice = *fcc* or *bcc* or N = # of neighbors per atom to include

Examples:

```
compute 1 all centro/atom fcc

compute 1 all centro/atom 8
```

Description:

Define a computation that calculates the centro−symmetry parameter for each atom in the group. In solid−state systems the centro−symmetry parameter is a useful measure of the local lattice disorder around an atom and can be used to characterize whether the atom is part of a perfect lattice, a local defect (e.g. a dislocation or stacking fault), or at a surface.

The value of the centro−symmetry parameter will be 0.0 for atoms not in the specified compute group.

This parameter is computed using the following formula from (Kelchner)

$$CS = \sum_{i=1}^{N/2} |\vec{R}_i + \vec{R}_{i+N/2}|^2$$

where the *N* nearest neighbors or each atom are identified and Ri and Ri+N/2 are vectors from the central atom to a particular pair of nearest neighbors. There are N*(N−1)/2 possible neighbor pairs that can contribute to this formula. The quantity in the sum is computed for each, and the N/2 smallest are used. This will typically be for pairs of atoms in symmetrically opposite positions with respect to the central atom; hence the i+N/2 notation.

N is an input parameter, which should be set to correspond to the number of nearest neighbors in the underlying lattice of atoms. If the keyword *fcc* or *bcc* is used, *N* is set to 12 and 8 respectively. More generally, *N* can be set to a positive, even integer.

For an atom on a lattice site, surrounded by atoms on a perfect lattice, the centro−symmetry parameter will be 0. It will be near 0 for small thermal perturbations of a perfect lattice. If a point defect exists, the symmetry is broken, and the parameter will be a larger positive value. An atom at a surface will have a large positive parameter. If the atom does not have *N* neighbors (within the potential cutoff), then its centro−symmetry parameter is set to 0.0.

Only atoms within the cutoff of the pairwise neighbor list are considered as possible neighbors. Atoms not in the compute group are included in the *N* neighbors used in this calculation.

The neighbor list needed to compute this quantity is constructed each time the calculation is performed (e.g. each

time a snapshot of atoms is dumped). Thus it can be inefficient to compute/dump this quantity too frequently or to have multiple compute/dump commands, each with a *centro/atom* style.

Output info:

This compute calculates a per−atom vector, which can be accessed by any command that uses per−atom values from a compute as input. See this section for an overview of LAMMPS output options.

The per−atom vector values are unitless values >= 0.0. Their magnitude depends on the lattice style due to the number of contibuting neighbor pairs in the summation in the formula above. And it depends on the local defects surrounding the central atom, as described above.

Here are typical centro−symmetry values, from a a nanoindentation simulation into gold (FCC). These were provided by Jon Zimmerman (Sandia):

```
Bulk lattice = 0
Dislocation core ~ 1.0 (0.5 to 1.25)
Stacking faults ~ 5.0 (4.0 to 6.0)
Free surface ~ 23.0
```

These values are *not* normalized by the square of the lattice parameter. If they were, normalized values would be:

```
Bulk lattice = 0
Dislocation core ~ 0.06 (0.03 to 0.075)
Stacking faults ~ 0.3 (0.24 to 0.36)
Free surface ~ 1.38
```

For BCC materials, the values for dislocation cores and free surfaces would be somewhat different, due to their being only 8 neighbors instead of 12.

Restrictions: none

Related commands:

compute cna/atom

Default: none

(Kelchner) Kelchner, Plimpton, Hamilton, Phys Rev B, 58, 11085 (1998).

compute cluster/atom command

Syntax:

```
compute ID group−ID cluster/atom cutoff
```

- ID, group−ID are documented in compute command
- cluster/atom = style name of this compute command
- cutoff = distance within which to label atoms as part of same cluster (distance units)

Examples:

```
compute 1 all cluster/atom 1.0
```

Description:

Define a computation that assigns each atom a cluster ID.

A cluster is defined as a set of atoms, each of which is within the cutoff distance from one or more other atoms in the cluster. If an atom has no neighbors within the cutoff distance, then it is a 1−atom cluster. The ID of every atom in the cluster will be the smallest atom ID of any atom in the cluster.

Only atoms in the compute group are clustered and assigned cluster IDs. Atoms not in the compute group are assigned a cluster ID = 0.

The neighbor list needed to compute this quantity is constructed each time the calculation is performed (i.e. each time a snapshot of atoms is dumped). Thus it can be inefficient to compute/dump this quantity too frequently or to have multiple compute/dump commands, each of a *clsuter/atom* style.

Output info:

This compute calculates a per−atom vector, which can be accessed by any command that uses per−atom values from a compute as input. See this section for an overview of LAMMPS output options.

The per−atom vector values will be an ID > 0, as explained above.

Restrictions: none

Related commands:

compute coord/atom

Default: none

compute cna/atom command

Syntax:

```
compute ID group-ID cna/atom cutoff
```

- ID, group−ID are documented in compute command
- cna/atom = style name of this compute command
- cutoff = cutoff distance for nearest neighbors (distance units)

Examples:

```
compute 1 all cna/atom 3.08
```

Description:

Define a computation that calculates the CNA (Common Neighbor Analysis) pattern for each atom in the group. In solid−state systems the CNA pattern is a useful measure of the local crystal structure around an atom. The CNA methodology is described in (Faken) and (Tsuzuki).

Currently, there are five kinds of CNA patterns LAMMPS recognizes:

- fcc = 1
- hcp = 2
- bcc = 3
- icosohedral = 4
- unknown = 5

The value of the CNA pattern will be 0 for atoms not in the specified compute group. Note that normally a CNA calculation should only be performed on mono−component systems.

The CNA calculation can be sensitive to the specified cutoff value. You should insure the appropriate nearest neighbors of an atom are found within the cutoff distance for the presumed crystal strucure. E.g. 12 nearest neighbor for perfect FCC and HCP crystals, 14 nearest neighbors for perfect BCC crystals. These formulas can be used to obtain a good cutoff distance:

$$
\begin{aligned}
r_c^{fcc} &= \frac{1}{2}\left(\frac{\sqrt{2}}{2}+1\right)\mathrm{a} \simeq 0.8536\,\mathrm{a} \\
r_c^{bcc} &= \frac{1}{2}(\sqrt{2}+1)\mathrm{a} \simeq 1.207\,\mathrm{a} \\
r_c^{hcp} &= \frac{1}{2}\left(1+\sqrt{\frac{4+2x^2}{3}}\right)\mathrm{a}
\end{aligned}
$$

where a is the lattice constant for the crystal structure concerned and in the HCP case, x = (c/a) / 1.633, where 1.633 is the ideal c/a for HCP crystals.

Also note that since the CNA calculation in LAMMPS uses the neighbors of an owned atom to find the nearest neighbors of a ghost atom, the following relation should also be satisfied:

$$Rc + Rs > 2 * \text{cutoff}$$

where Rc is the cutoff distance of the potential, Rs is the skin distance as specified by the neighbor command, and cutoff is the argument used with the compute cna/atom command. LAMMPS will issue a warning if this is not the case.

The neighbor list needed to compute this quantity is constructed each time the calculation is performed (e.g. each time a snapshot of atoms is dumped). Thus it can be inefficient to compute/dump this quantity too frequently or to have multiple compute/dump commands, each with a *cna/atom* style.

Output info:

This compute calculates a per−atom vector, which can be accessed by any command that uses per−atom values from a compute as input. See this section for an overview of LAMMPS output options.

The per−atom vector values will be a number from 0 to 5, as explained above.

Restrictions: none

Related commands:

compute centro/atom

Default: none

(Faken) Faken, Jonsson, Comput Mater Sci, 2, 279 (1994).

(Tsuzuki) Tsuzuki, Branicio, Rino, Comput Phys Comm, 177, 518 (2007).

compute com command

Syntax:

```
compute ID group-ID com
```

- ID, group−ID are documented in compute command
- com = style name of this compute command

Examples:

```
compute 1 all com
```

Description:

Define a computation that calculates the center−of−mass of the group of atoms, including all effects due to atoms passing thru periodic boundaries.

A vector of three quantites is calculated by this compute, which are the x,y,z coordinates of the center of mass.

IMPORTANT NOTE: The coordinates of an atom contribute to the center−of−mass in "unwrapped" form, by using the image flags associated with each atom. See the dump custom command for a discussion of "unwrapped" coordinates. See the Atoms section of the read_data command for a discussion of image flags and how they are set for each atom. You can reset the image flags (e.g. to 0) before invoking this compute by using the set image command.

IMPORTANT NOTE: If an atom is part of a rigid body (see the fix rigid command), it's periodic image flags are altered, and its contribution to the center−of−mass may not reflect its true contribution. See the fix rigid command for details. Thus, to compute the center−of−mass of rigid bodies as they cross periodic boundaries, you will need to post−process a dump file containing coordinates of the atoms in the bodies.

Output info:

This compute calculates a global vector of length 3, which can be accessed by indices 1−3 by any command that uses global vector values from a compute as input. See this section for an overview of LAMMPS output options.

The vector values are "intensive". The vector values will be in distance units.

Restrictions: none

Related commands:

compute com/molecule

Default: none

compute com/molecule command

Syntax:

```
compute ID group-ID com/molecule
```

- ID, group−ID are documented in compute command
- com/molecule = style name of this compute command

Examples:

```
compute 1 fluid com/molecule
```

Description:

Define a computation that calculates the center−of−mass of individual molecules. The calculation includes all effects due to atoms passing thru periodic boundaries.

The x,y,z coordinates of the center−of−mass for a particular molecule are only computed if one or more of its atoms are in the specified group. Normally all atoms in the molecule should be in the group, however this is not required. LAMMPS will warn you if this is not the case. Only atoms in the group contribute to the center−of−mass calculation for the molecule.

The ordering of per−molecule quantities produced by this compute is consistent with the ordering produced by other compute commands that generate per−molecule datums. Conceptually, them molecule IDs will be in ascending order for any molecule with one or more of its atoms in the specified group.

IMPORTANT NOTE: The coordinates of an atom contribute to the molecule's center−of−mass in "unwrapped" form, by using the image flags associated with each atom. See the dump custom command for a discussion of "unwrapped" coordinates. See the Atoms section of the read_data command for a discussion of image flags and how they are set for each atom. You can reset the image flags (e.g. to 0) before invoking this compute by using the set image command.

IMPORTANT NOTE: If an atom is part of a rigid body (see the fix rigid command), it's periodic image flags are altered, and its contribution to the center−of−mass may not reflect its true contribution. See the fix rigid command for details. Thus, to compute the center−of−mass of rigid bodies as they cross periodic boundaries, you will need to post−process a dump file containing coordinates of the atoms in the bodies.

Output info:

This compute calculates a global array where the number of rows = Nmolecules and the number of columns = 3 for the x,y,z center−of−mass coordinates of each molecule. These values can be accessed by any command that uses global array values from a compute as input. See this section for an overview of LAMMPS output options.

The array values are "intensive". The array values will be in distance units.

Restrictions: none

Related commands:

compute com

Default: none

compute coord/atom command

Syntax:

```
compute ID group−ID coord/atom cutoff
```

- ID, group−ID are documented in compute command
- coord/atom = style name of this compute command
- cutoff = distance within which to count coordination neighbors (distance units)

Examples:

```
compute 1 all coord/atom 2.0
```

Description:

Define a computation that calculates the coordination number for each atom in a group.

The value of the coordination number will be 0.0 for atoms not in the specified compute group.

The coordination number is defined as the number of neighbor atoms within the specified cutoff distance from the central atom. Atoms not in the group are included in the coordination number of atoms in the group.

The neighbor list needed to compute this quantity is constructed each time the calculation is performed (i.e. each time a snapshot of atoms is dumped). Thus it can be inefficient to compute/dump this quantity too frequently or to have multiple compute/dump commands, each of a *coord/atom* style.

Output info:

This compute calculates a per−atom vector, which can be accessed by any command that uses per−atom values from a compute as input. See this section for an overview of LAMMPS output options.

The per−atom vector values will be a number >= 0.0, as explained above.

Restrictions: none

Related commands:

compute cluster/atom

Default: none

compute damage/atom command

Syntax:

```
compute ID group-ID damage/atom
```

- ID, group−ID are documented in compute command
- damage/atom = style name of this compute command

Examples:

```
compute 1 all damage/atom
```

Description:

Define a computation that calculates the per−atom damage for each atom in a group. Please see the PDLAMMPS user guide for a formal definition of "damage" and more details about Peridynamics as it is implemented in LAMMPS.

The value of the damage will be 0.0 for atoms not in the specified compute group.

Output info:

This compute calculates a per−atom vector, which can be accessed by any command that uses per−atom values from a compute as input. See this section for an overview of LAMMPS output options.

The per−atom vector values will be a number >= 0.0, as explained above.

Restrictions:

This compute is part of the "peri" package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

Related commands:

dump custom

Default: none

compute dihedral/local command

Syntax:

```
compute ID group-ID dihedral/local input1 input2 ...
```

- ID, group−ID are documented in compute command
- dihedral/local = style name of this compute command
- zero or more keywords may be appended
- keyword = *phi*

  phi = tabulate dihedral angles

Examples:

```
compute 1 all dihedral/local phi
```

Description:

Define a computation that calculates properties of individual dihedral interactions. The number of datums generated, aggregated across all processors, equals the number of angles in the system.

The local data stored by this command is generated by looping over all the atoms owned on a processor and their dihedrals. A dihedral will only be included if all 4 atoms in the dihedral are in the specified compute group.

Note that as atoms migrate from processor to processor, there will be no consistent ordering of the entries within the local vector or array from one timestep to the next. The only consistency that is guaranteed is that the ordering on a particular timestep will be the same for local vectors or arrays generated by other compute commands. For example, dihedral output from the compute property/local command can be combined with data from this command and output by the dump local command in a consistent way.

Output info:

This compute calculates a local vector or local array depending on the number of keywords. The length of the vector or number of rows in the array is the number of dihedrals. If a single keyword is specified, a local vector is produced. If two or more keywords are specified, a local array is produced where the number of columns = the number of keywords. The vector or array can be accessed by any command that uses local values from a compute as input. See this section for an overview of LAMMPS output options.

The output for *phi* will be in degrees.

Restrictions: none

Related commands:

dump local, compute property/local

Default: none

compute displace/atom command

Syntax:

```
compute ID group-ID displace/atom
```

- ID, group−ID are documented in compute command
- displace/atom = style name of this compute command

Examples:

```
compute 1 all displace/atom
```

Description:

Define a computation that calculates the current displacement of each atom in the group from its original coordinates, including all effects due to atoms passing thru periodic boundaries.

A vector of four quantites per atom is calculated by this compute. The first 3 elements of the vector are the dx,dy,dz displacements. The 4th component is the total displacement, i.e. sqrt(dx*dx + dy*dy + dz*dz).

The displacement of an atom is from its original position at the time the compute command was issued. To store the original coordinates, the compute creates its own fix of style "store/state", as if this command had been issued:

```
fix compute-ID_store_state group-ID store/state xu yu zu
```

See the fix store/state command for details. Note that the ID of the new fix is the compute−ID + underscore + "store/state", and the group for the new fix is the same as the compute group.

The value of the displacement will be 0.0 for atoms not in the specified compute group.

IMPORTANT NOTE: Fix store/state stores the initial coordinates in "unwrapped" form, by using the image flags associated with each atom. See the dump custom command for a discussion of "unwrapped" coordinates. See the Atoms section of the read_data command for a discussion of image flags and how they are set for each atom. You can reset the image flags (e.g. to 0) before invoking this compute by using the set image command.

IMPORTANT NOTE: If an atom is part of a rigid body (see the fix rigid command), it's periodic image flags are altered, and the computed displacement may not reflect its true displacement. See the fix rigid command for details. Thus, to compute the displacement of rigid bodies as they cross periodic boundaries, you will need to post−process a dump file containing coordinates of the atoms in the bodies.

IMPORTANT NOTE: If you want the quantities calculated by this compute to be continuous when running from a restart file, then you should use the same ID for this compute, as in the original run. This is so that the created fix will also have the same ID, and thus be initialized correctly with atom coordinates from the restart file.

Output info:

This compute calculates a per−atom array with 4 columns, which can be accessed by indices 1−4 by any command that uses per−atom values from a compute as input. See this section for an overview of LAMMPS output options.

The per−atom array values will be in distance units.

Restrictions: none

Related commands:

compute msd, dump custom, fix store/state

Default: none

compute erotate/asphere command

Syntax:

```
compute ID group-ID erotate/asphere
```

- ID, group−ID are documented in compute command
- erotate/asphere = style name of this compute command

Examples:

```
compute 1 all erotate/asphere
```

Description:

Define a computation that calculates the rotational kinetic energy of a group of aspherical particles.

The rotational kinetic energy is computed as 1/2 I w^2, where I is the inertia tensor for the aspherical particle and w is its angular velocity, which is computed from its angular momentum.

IMPORTANT NOTE: For 2d models, particles are treated as ellipsoids, not ellipses, meaning their moments of inertia will be the same as in 3d.

Output info:

This compute calculates a global scalar (the KE). This value can be used by any command that uses a global scalar value from a compute as input. See this section for an overview of LAMMPS output options.

The scalar value calculated by this compute is "extensive". The scalar value will be in energy units.

Restrictions:

This compute requires that atoms store a shape and quaternion orientation and angular momentum as defined by the atom_style ellipsoid command.

All particles in the group must be finite−size. They cannot be point particles.

Related commands: none

compute erotate/sphere

Default: none

compute erotate/sphere command

Syntax:

```
compute ID group-ID erotate/sphere
```

- ID, group−ID are documented in compute command
- erotate/sphere = style name of this compute command

Examples:

```
compute 1 all erotate/sphere
```

Description:

Define a computation that calculates the rotational kinetic energy of a group of spherical particles.

The rotational energy is computed as 1/2 I w^2, where I is the moment of inertia for a sphere and w is the particle's angular velocity.

IMPORTANT NOTE: For 2d models, particles are treated as spheres, not disks, meaning their moment of inertia will be the same as in 3d.

Output info:

This compute calculates a global scalar (the KE). This value can be used by any command that uses a global scalar value from a compute as input. See this section for an overview of LAMMPS output options.

The scalar value calculated by this compute is "extensive". The scalar value will be in energy units.

Restrictions:

This compute requires that atoms store a radius and angular velocity (omega) as defined by the atom_style sphere command.

All particles in the group must be finite−size spheres or point particles. They cannot be aspherical. Point particles will not contribute to the rotational energy.

Related commands:

compute erotate/asphere

Default: none

compute event/displace command

Syntax:

```
compute ID group-ID event/displace threshold
```

- ID, group−ID are documented in compute command
- event/displace = style name of this compute command
- threshold = minimum distance anyparticle must move to trigger an event (distance units)

Examples:

```
compute 1 all event/displace 0.5
```

Description:

Define a computation that flags an "event" if any particle in the group has moved a distance greater than the specified threshold distance when compared to a previously stored reference state (i.e. the previous event). This compute is typically used in conjunction with the prd and tad commands, to detect if a transition to a new minimum energy basin has occurred.

This value calculated by the compute is equal to 0 if no particle has moved far enough, and equal to 1 if one or more particles have moved further than the threshold distance.

NOTE: If the system is undergoing significant center−of−mass motion, due to thermal motion, an external force, or an initial net momentum, then this compute will not be able to distinguish that motion from local atom displacements and may generate "false postives."

Output info:

This compute calculates a global scalar (the flag). This value can be used by any command that uses a global scalar value from a compute as input. See this section for an overview of LAMMPS output options.

The scalar value calculated by this compute is "intensive". The scalar value will be a 0 or 1 as explained above.

Restrictions:

This command can only be used if LAMMPS was built with the "replica" package. See the Making LAMMPS section for more info on packages.

Related commands:

prd, tad

Default: none

compute group/group command

Syntax:

```
compute ID group-ID group/group group2-ID
```

- ID, group−ID are documented in compute command
- group/group = style name of this compute command
- group2−ID = group ID of second (or same) group

Examples:

```
compute 1 lower group/group upper
compute mine fluid group/group wall
```

Description:

Define a computation that calculates the total energy and force interaction between two groups of atoms: the compute group and the specified group2. The two groups can be the same. The interaction energy is defined as the pairwise energy between all pairs of atoms where one atom in the pair is in the first group and the other is in the second group. Likewise, the interaction force calculated by this compute is the force on the compute group atoms due to pairwise interactions with atoms in the specified group2.

The energy and force are calculated by looping over a neighbor list of pairwise interactions. Thus it can be inefficient to compute this quantity too frequently.

Output info:

This compute calculates a global scalar (the energy) and a global vector of length 3 (force), which can be accessed by indices 1−3. These values can be used by any command that uses global scalar or vector values from a compute as input. See this section for an overview of LAMMPS output options.

Both the scalar and vector values calculated by this compute are "extensive". The scalar value will be in energy units. The vector values will be in force units.

Restrictions:

Only pairwise interactions, as defined by the pair_style command, are included in this calculation. Bond (angle, dihedral, etc) interactions between atoms in the two groups are not included. Long−range interactions due to a kspace_style command are also not included. Not all pair potentials can be evaluated in a pairwise mode as required by this compute. For example, 3−body potentials, such as Tersoff and Stillinger−Weber cannot be used. EAM potentials for metals only include the pair potential portion of the EAM interaction, not the embedding term.

Related commands: none

Default: none

compute gyration command

Syntax:

```
compute ID group-ID gyration
```

- ID, group−ID are documented in compute command
- gyration = style name of this compute command

Examples:

```
compute 1 molecule gyration
```

Description:

Define a computation that calculates the radius of gyration Rg of the group of atoms, including all effects due to atoms passing thru periodic boundaries.

Rg is a measure of the size of the group of atoms, and is computed by this formula

$$R_g{}^2 = \frac{1}{M} \sum_i m_i (r_i - r_{cm})^2$$

where M is the total mass of the group, Rcm is the center−of−mass position of the group, and the sum is over all atoms in the group.

IMPORTANT NOTE: The coordinates of an atom contribute to Rg in "unwrapped" form, by using the image flags associated with each atom. See the dump custom command for a discussion of "unwrapped" coordinates. See the Atoms section of the read_data command for a discussion of image flags and how they are set for each atom. You can reset the image flags (e.g. to 0) before invoking this compute by using the set image command.

Output info:

This compute calculates a global scalar (Rg). This value can be used by any command that uses a global scalar value from a compute as input. See this section for an overview of LAMMPS output options.

The scalar value calculated by this compute is "intensive". The scalar value will be in distance units.

Restrictions: none

Related commands:

compute gyration/molecule

Default: none

compute gyration/molecule command

Syntax:

```
compute ID group-ID gyration/molecule
```

- ID, group−ID are documented in compute command
- gyration/molecule = style name of this compute command

Examples:

```
compute 1 molecule gyration/molecule
```

Description:

Define a computation that calculates the radius of gyration Rg of individual molecules. The calculation includes all effects due to atoms passing thru periodic boundaries.

Rg is a measure of the size of a molecule, and is computed by this formula

$$R_g{}^2 = \frac{1}{M} \sum_i m_i (r_i - r_{cm})^2$$

where M is the total mass of the molecule, Rcm is the center−of−mass position of the molecule, and the sum is over all atoms in the molecule and in the group.

Rg for a particular molecule is only computed if one or more of its atoms are in the specified group. Normally all atoms in the molecule should be in the group, however this is not required. LAMMPS will warn you if this is not the case. Only atoms in the group contribute to the Rg calculation for the molecule.

The ordering of per−molecule quantities produced by this compute is consistent with the ordering produced by other compute commands that generate per−molecule datums. Conceptually, them molecule IDs will be in ascending order for any molecule with one or more of its atoms in the specified group.

IMPORTANT NOTE: The coordinates of an atom contribute to Rg in "unwrapped" form, by using the image flags associated with each atom. See the dump custom command for a discussion of "unwrapped" coordinates. See the Atoms section of the read_data command for a discussion of image flags and how they are set for each atom. You can reset the image flags (e.g. to 0) before invoking this compute by using the set image command.

Output info:

This compute calculates a global vector of Rg values where the length of the vector = Nmolecules. These values can be used by any command that uses global vector values from a compute as input. See this section for an overview of LAMMPS output options.

The vector values calculated by this compute are "intensive". The vector values will be in distance units.

Restrictions: none

Related commands: none

compute gyration

Default: none

compute heat/flux command

Syntax:

```
compute ID group-ID heat/flux ke-ID pe-ID stress-ID
```

- ID, group−ID are documented in compute command
- heat/flux = style name of this compute command
- ke−ID = ID of a compute that calculates per−atom kinetic energy
- pe−ID = ID of a compute that calculates per−atom potential energy
- stress−ID = ID of a compute that calculates per−atom stress

Examples:

```
compute myFlux all heat/flux myKE myPE myStress
```

Description:

Define a computation that calculates the heat flux vector based on contributions from atoms in the specified group. This can be used by itself to measure the heat flux into or out of a reservoir of atoms, or to calculate a thermal conductivity using the Green−Kubo formalism.

See the fix thermal/conductivity command for details on how to compute thermal conductivity in an alternate way, via the Muller−Plathe method. See the fix heat command for a way to control the heat added or subtracted to a group of atoms.

The compute takes three arguments which are IDs of other computes. One calculates per−atom kinetic energy (*ke−ID*), one calculates per−atom potential energy (*pe−ID)*, and the third calcualtes per−atom stress (*stress−ID*). These should be defined for the same group used by compute heat/flux, though LAMMPS does not check for this.

The Green−Kubo formulas relate the ensemble average of the auto−correlation of the heat flux J to the thermal conductivity kappa:

$$\begin{aligned}\mathbf{J} &= \frac{1}{V}\left[\sum_i e_i\mathbf{v}_i - \sum_i \mathbf{S}_i\mathbf{v}_i\right] \\ &= \frac{1}{V}\left[\sum_i e_i\mathbf{v}_i + \sum_{i<j}(\mathbf{f}_{ij}\cdot\mathbf{v}_j)\,\mathbf{x}_{ij}\right] \\ &= \frac{1}{V}\left[\sum_i e_i\mathbf{v}_i + \frac{1}{2}\sum_{i<j}(\mathbf{f}_{ij}\cdot(\mathbf{v}_i+\mathbf{v}_j))\,\mathbf{x}_{ij}\right]\end{aligned}$$

$$\kappa = \frac{V}{k_B T^2}\int_0^\infty \langle J_x(0)\,J_x(t)\rangle\,dt = \frac{V}{3k_B T^2}\int_0^\infty \langle \mathbf{J}(0)\cdot\mathbf{J}(t)\rangle\,dt$$

Ei in the first term of the equation for J is the per−atom energy (potential and kinetic). This is calculated by the computes *ke−ID* and *pe−ID*. Si in the second term of the equation for J is the per−atom stress tensor calculated by the compute *stress−ID*. The tensor multiplies Vi as a 3x3 matrix−vector multiply to yield a vector. Note that as discussed below, the 1/V scaling factor in the equation for J is NOT included in the calculation performed by this compute; you need to add it for a volume appropriate to the atoms included in the calculation.

IMPORTANT NOTE: The compute pe/atom and compute stress/atom commands have options for which terms to include in their calculation (pair, bond, etc). The heat flux calculation will thus include exactly the same terms. Normally you should use compute stress/atom virial so as not to include a kinetic energy term in the heat flux. Note that neither of those computes is able to include a long−range Coulombic contribution to the per−atom energy or stress.

This compute calculates 6 quantities and stores them in a 6−component vector. The first 3 components are the x, y, z components of the full heat flux vector, i.e. (Jx, Jy, Jz). The next 3 components are the x, y, z components of just the convective portion of the flux, i.e. the first term in the equation for J above.

The heat flux can be output every so many timesteps (e.g. via the thermo_style custom command). Then as a post−processing operation, an autocorrelation can be performed, its integral estimated, and the Green−Kubo formula above evaluated.

The fix ave/correlate command can calclate the autocorrelation. The trap() function in the variable command can calculate the integral.

An example LAMMPS input script for solid Ar is appended below. The result should be: average conductivity ~0.29 in W/mK.

Output info:

This compute calculates a global vector of length 6 (total heat flux vector, followed by conductive heat flux vector), which can be accessed by indices 1−6. These values can be used by any command that uses global vector values from a compute as input. See this section for an overview of LAMMPS output options.

The vector values calculated by this compute are "extensive", meaning they scale with the number of atoms in the simulation. They can be divided by the appropriate volume to get a flux, which would then be an "intensive" value, meaning independent of the number of atoms in the simulation. Note that if the compute is "all", then the appropriate volume to divide by is the simulation box volume. However, if a sub−group is used, it should be the volume containing those atoms.

The vector values will be in energy*velocity units. Once divided by a volume the units will be that of flux, namely energy/area/time units

Restrictions: none

Related commands:

fix thermal/conductivity, fix ave/correlate, variable

Default: none

```
# Sample LAMMPS input script for thermal conductivity of solid Ar

units       real
variable    T equal 70
```

```
variable    V equal vol
variable    dt equal 4.0
variable    p equal 200     # correlation length
variable    s equal 10      # sample interval
variable    d equal $p*$s   # dump interval

# convert from LAMMPS real units to SI

variable    kB equal 1.3806504e-23    # [J/K] Boltzmann
variable    kCal2J equal 4186.0/6.02214e23
variable    A2m equal 1.0e-10
variable    fs2s equal 1.0e-15
variable    convert equal ${kCal2J}*${kCal2J}/${fs2s}/${A2m}

# setup problem

dimension    3
boundary     p p p
lattice      fcc 5.376 orient x 1 0 0 orient y 0 1 0 orient z 0 0 1
region       box block 0 4 0 4 0 4
create_box   1 box
create_atoms 1 box
mass         1 39.948
pair_style   lj/cut 13.0
pair_coeff   * * 0.2381 3.405
timestep     ${dt}
thermo       $d

# equilibration and thermalization

velocity     all create $T 102486 mom yes rot yes dist gaussian
fix          NVT all nvt temp $T $T 10 drag 0.2
run          8000

# thermal conductivity calculation, switch to NVE if desired

#unfix       NVT
#fix         NVE all nve

reset_timestep 0
compute      myKE all ke/atom
compute      myPE all pe/atom
compute      myStress all stress/atom virial
compute      flux all heat/flux myKE myPE myStress
variable     Jx equal c_flux[1]/vol
variable     Jy equal c_flux[2]/vol
variable     Jz equal c_flux[3]/vol
fix          JJ all ave/correlate $s $p $d &             c_flux[1] c_flux[2] c_flux[3] type auto fil
variable     scale equal ${convert}/${kB}/$T/$T/$V*$s*${dt}
variable     k11 equal trap(f_JJ[3])*${scale}
variable     k22 equal trap(f_JJ[4])*${scale}
variable     k33 equal trap(f_JJ[5])*${scale}
thermo_style custom step temp v_Jx v_Jy v_Jz v_k11 v_k22 v_k33
run          100000
variable     k equal (v_k11+v_k22+v_k33)/3.0
variable     ndens equal count(all)/vol
print        "average conductivity: $k[W/mK] @ $T K, ${ndens} /A^3"
```

compute improper/local command

Syntax:

```
compute ID group-ID improper/local input1 input2 ...
```

- ID, group−ID are documented in compute command
- improper/local = style name of this compute command
- zero or more keywords may be appended
- keyword = *chi*

```
  chi = tabulate improper angles
```

Examples:

```
compute 1 all improper/local chi
```

Description:

Define a computation that calculates properties of individual improper interactions. The number of datums generated, aggregated across all processors, equals the number of impropers in the system.

The local data stored by this command is generated by looping over all the atoms owned on a processor and their impropers. An improper will only be included if all 4 atoms in the improper are in the specified compute group.

Note that as atoms migrate from processor to processor, there will be no consistent ordering of the entries within the local vector or array from one timestep to the next. The only consistency that is guaranteed is that the ordering on a particular timestep will be the same for local vectors or arrays generated by other compute commands. For example, improper output from the compute property/local command can be combined with data from this command and output by the dump local command in a consistent way.

Output info:

This compute calculates a local vector or local array depending on the number of keywords. The length of the vector or number of rows in the array is the number of impropers. If a single keyword is specified, a local vector is produced. If two or more keywords are specified, a local array is produced where the number of columns = the number of keywords. The vector or array can be accessed by any command that uses local values from a compute as input. See this section for an overview of LAMMPS output options.

The output for *chi* will be in degrees.

Restrictions: none

Related commands:

dump local, compute property/local

Default: none

compute ke command

Syntax:

```
compute ID group-ID ke
```

- ID, group−ID are documented in compute command
- ke = style name of this compute command

Examples:

```
compute 1 all ke
```

Description:

Define a computation that calculates the translational kinetic energy of a group of particles.

The kinetic energy or each particle is computed as 1/2 m v^2, where m and v are the mass and velocity of the particle.

There is a subtle difference between the quantity calculated by this compute and the kinetic energy calculated by the *ke* or *etotal* keyword used in thermodynamic output, as specified by the thermo_style command. For this compute, kinetic energy is "translational" kinetic energy, calculated by the simple formula above. For thermodynamic output, the *ke* keyword infers kinetic energy from the temperature of the system with 1/2 Kb T of energy for each degree of freedom. For the default temperature computation via the compute temp command, these are the same. But different computes that calculate temperature can subtract out different non−thermal components of velocity and/or include different degrees of freedom (translational, rotational, etc).

Output info:

This compute calculates a global scalar (the KE). This value can be used by any command that uses a global scalar value from a compute as input. See this section for an overview of LAMMPS output options.

The scalar value calculated by this compute is "extensive". The scalar value will be in energy units.

Restrictions: none

Related commands:

compute erotate/sphere

Default: none

compute ke/atom command

Syntax:

```
compute ID group−ID ke/atom
```

- ID, group−ID are documented in compute command
- ke/atom = style name of this compute command

Examples:

```
compute 1 all ke/atom
```

Description:

Define a computation that calculates the per−atom translational kinetic energy for each atom in a group.

The kinetic energy is simply 1/2 m v^2, where m is the mass and v is the velocity of each atom.

The value of the kinetic energy will be 0.0 for atoms not in the specified compute group.

Output info:

This compute calculates a per−atom vector, which can be accessed by any command that uses per−atom values from a compute as input. See this section for an overview of LAMMPS output options.

The per−atom vector values will be in energy units.

Restrictions: none

Related commands:

dump custom

Default: none

compute ke/atom/eff command

Syntax:

```
compute ID group-ID ke/atom/eff
```

- ID, group−ID are documented in compute command
- ke/atom/eff = style name of this compute command

Examples:

```
compute 1 all ke/atom/eff
```

Description:

Define a computation that calculates the per−atom translational (nuclei and electrons) and radial kinetic energy (electron only) in a group. The particles are assumed to be nuclei and electrons modeled with the electronic force field.

The kinetic energy for each nucleus is computed as 1/2 m v^2, where m corresponds to the corresponding nuclear mass, and the kinetic energy for each electron is computed as 1/2 (me v^2 + 3/4 me s^2), where me and v correspond to the mass and translational velocity of each electron, and s to its radial velocity, respectively.

There is a subtle difference between the quantity calculated by this compute and the kinetic energy calculated by the *ke* or *etotal* keyword used in thermodynamic output, as specified by the thermo_style command. For this compute, kinetic energy is "translational" plus electronic "radial" kinetic energy, calculated by the simple formula above. For thermodynamic output, the *ke* keyword infers kinetic energy from the temperature of the system with 1/2 Kb T of energy for each (nuclear−only) degree of freedom in eFF.

IMPORTANT NOTE: The temperature in eFF should be monitored via the compute temp/eff command, which can be printed with thermodynamic output by using the thermo_modify command, as shown in the following example:

```
compute         effTemp all temp/eff
thermo_style    custom step etotal pe ke temp press
thermo_modify   temp effTemp
```

The value of the kinetic energy will be 0.0 for atoms (nuclei or electrons) not in the specified compute group.

Output info:

This compute calculates a scalar quantity for each atom, which can be accessed by any command that uses per−atom computes as input. See this section for an overview of LAMMPS output options.

The per−atom vector values will be in energy units.

Restrictions:

This compute is part of the "user−eff" package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

Related commands:

dump custom

Default: none

compute ke/eff command

Syntax:

```
compute ID group-ID ke/eff
```

- ID, group−ID are documented in compute command
- ke/eff = style name of this compute command

Examples:

```
compute 1 all ke/eff
```

Description:

Define a computation that calculates the kinetic energy of motion of a group of eFF particles (nuclei and electrons), as modeled with the electronic force field.

The kinetic energy for each nucleus is computed as 1/2 m v^2 and the kinetic energy for each electron is computed as 1/2(me v^2 + 3/4 me s^2), where m corresponds to the nuclear mass, me to the electron mass, v to the translational velocity of each particle, and s to the radial velocity of the electron, respectively.

There is a subtle difference between the quantity calculated by this compute and the kinetic energy calculated by the *ke* or *etotal* keyword used in thermodynamic output, as specified by the thermo_style command. For this compute, kinetic energy is "translational" and "radial" (only for electrons) kinetic energy, calculated by the simple formula above. For thermodynamic output, the *ke* keyword infers kinetic energy from the temperature of the system with 1/2 Kb T of energy for each degree of freedom. For the eFF temperature computation via the compute temp_eff command, these are the same. But different computes that calculate temperature can subtract out different non−thermal components of velocity and/or include other degrees of freedom.

IMPRORTANT NOTE: The temperature in eFF models should be monitored via the compute temp/eff command, which can be printed with thermodynamic output by using the thermo_modify command, as shown in the following example:

```
compute         effTemp all temp/eff
thermo_style    custom step etotal pe ke temp press
thermo_modify   temp effTemp
```

See compute temp/eff.

Output info:

This compute calculates a global scalar (the KE). This value can be used by any command that uses a global scalar value from a compute as input. See this section for an overview of LAMMPS output options.

The scalar value calculated by this compute is "extensive". The scalar value will be in energy units.

Restrictions:

This compute is part of the "user−eff" package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

Related commands: none

Default: none

compute_modify command

Syntax:

```
compute_modify compute-ID keyword value ...
```

- compute-ID = ID of the compute to modify
- one or more keyword/value pairs may be listed
- keyword = *extra* or *dynamic*

```
  extra value = N
      N = # of extra degrees of freedom to subtract
    dynamic value = yes or no
      yes/no = do or do not recompute the number of atoms contributing to the temperature
    thermo value = yes or no
      yes/no = do or do not add contributions from fixes to the potential energy
```

Examples:

```
compute_modify myTemp extra 0
compute_modify newtemp dynamic yes extra 600
```

Description:

Modify one or more parameters of a previously defined compute. Not all compute styles support all parameters.

The *extra* keyword refers to how many degrees−of−freedom are subtracted (typically from 3N) as a normalizing factor in a temperature computation. Only computes that compute a temperature use this option. The default is 2 or 3 for 2d or 3d systems which is a correction factor for an ensemble of velocities with zero total linear momentum. You can use a negative number for the *extra* parameter if you need to add degrees−of−freedom. See the compute temp/asphere command for an example.

The *dynamic* keyword determines whether the number of atoms N in the compute group is re−computed each time a temperature is computed. Only compute styles that compute a temperature use this option. By default, N is assumed to be constant. If you are adding atoms to the system (see the fix pour or fix deposit commands) or expect atoms to be lost (e.g. due to evaporation), then this option can be used to insure the temperature is correctly normalized.

The *thermo* keyword determines whether the potential energy contribution calculated by some fixes is added to the potential energy calculated by the compute. Currently, only the compute of style *pe* uses this option. See the doc pages for individual fixes for details.

Restrictions: none

Related commands:

compute

Default:

The option defaults are extra = 2 or 3 for 2d or 3d systems and dynamic = no. Thermo is *yes* if the compute of style *pe* was defined with no extra keywords; otherwise it is *no*.

compute msd command

Syntax:

```
compute ID group-ID msd keyword values ...
```

- ID, group−ID are documented in compute command
- msd = style name of this compute command
- zero or more keyword/value pairs may be appended
- keyword = *com*

  com value = *yes* or *no*

Examples:

```
compute 1 all msd
compute 1 upper msd com yes
```

Description:

Define a computation that calculates the mean−squared displacement (MSD) of the group of atoms, including all effects due to atoms passing thru periodic boundaries.

A vector of four quantites is calculated by this compute. The first 3 elements of the vector are the squared dx,dy,dz displacements, summed and averaged over atoms in the group. The 4th component is the total squared displacement, i.e. (dx*dx + dy*dy + dz*dz), summed and averaged over atoms in the group.

The slope of the mean−squared displacement (MSD) versus time is proportional to the diffusion coefficient of the diffusing atoms.

The displacement of an atom is from its original position at the time the compute command was issued. To store the original coordinates, the compute creates its own fix of style "store/state", as if this command had been issued:

```
fix compute-ID_store_state group-ID store/state xu yu zu
```

See the fix store/state command for details. Note that the ID of the new fix is the compute−ID + underscore + "store_state", and the group for the new fix is the same as the compute group.

If the *com* option is set to *yes* then the effect of any drift in the center−of−mass of the group of atoms is subtracted out before the displacment of each atom is calcluated. The *com* option is also passed to the created fix store/state.

IMPORTANT NOTE: Fix store/state stores the initial coordinates in "unwrapped" form, by using the image flags associated with each atom. See the dump custom command for a discussion of "unwrapped" coordinates. See the Atoms section of the read_data command for a discussion of image flags and how they are set for each atom. You can reset the image flags (e.g. to 0) before invoking this compute by using the set image command.

IMPORTANT NOTE: If an atom is part of a rigid body (see the fix rigid command), it's periodic image flags are altered, and its contribution to the MSD may not reflect its true contribution. See the fix rigid command for details. Thus, to compute the MSD of rigid bodies as they cross periodic boundaries, you will need to post−process a dump file containing coordinates of the atoms in the bodies.

IMPORTANT NOTE: If you want the quantities calculated by this compute to be continuous when running from a restart file, then you should use the same ID for this compute, as in the original run. This is so that the created fix will also have the same ID, and thus be initialized correctly with atom coordinates from the restart file.

Output info:

This compute calculates a global vector of length 4, which can be accessed by indices 1−4 by any command that uses global vector values from a compute as input. See this section for an overview of LAMMPS output options.

The vector values are "intensive". The vector values will be in distance^2 units.

Restrictions: none

Related commands:

compute displace_atom, fix store/state, compute msd/molecule

Default:

The option default is com = no.

compute msd/molecule command

Syntax:

```
compute ID group-ID msd/molecule
```

- ID, group−ID are documented in compute command
- msd/molecule = style name of this compute command

Examples:

```
compute 1 all msd/molecule
```

Description:

Define a computation that calculates the mean−squared displacement (MSD) of individual molecules. The calculation includes all effects due to atoms passing thru periodic boundaries.

Four quantites are calculated by this compute for each molecule. The first 3 quantities are the squared dx,dy,dz displacements of the center−of−mass. The 4th component is the total squared displacement, i.e. (dx*dx + dy*dy + dz*dz) of the center−of−mass.

The slope of the mean−squared displacement (MSD) versus time is proportional to the diffusion coefficient of the diffusing molecules.

The displacement of the center−of−mass of the molecule is from its original center−of−mass position at the time the compute command was issued.

The MSD for a particular molecule is only computed if one or more of its atoms are in the specified group. Normally all atoms in the molecule should be in the group, however this is not required. LAMMPS will warn you if this is not the case. Only atoms in the group contribute to the center−of−mass calculation for the molecule, which is used to caculate its initial and current position.

The ordering of per−molecule quantities produced by this compute is consistent with the ordering produced by other compute commands that generate per−molecule datums. Conceptually, them molecule IDs will be in ascending order for any molecule with one or more of its atoms in the specified group.

IMPORTANT NOTE: The initial coordinates of each molecule are stored in "unwrapped" form, by using the image flags associated with each atom. See the dump custom command for a discussion of "unwrapped" coordinates. See the Atoms section of the read_data command for a discussion of image flags and how they are set for each atom. You can reset the image flags (e.g. to 0) before invoking this compute by using the set image command.

IMPORTANT NOTE: If an atom is part of a rigid body (see the fix rigid command), it's periodic image flags are altered, and its contribution to the MSD may not reflect its true contribution. See the fix rigid command for details. Thus, to compute the MSD of rigid bodies as they cross periodic boundaries, you will need to post−process a dump file containing coordinates of the atoms in the bodies.

IMPORTANT NOTE: Unlike the compute msd command, this compute does not store the initial center−of−mass coorindates of its molecules in a restart file. Thus you cannot continue the MSD per molecule calculation of this

compute when running from a restart file.

Output info:

This compute calculates a global array where the number of rows = Nmolecules and the number of columns = 4 for dx,dy,dz and the total displacement. These values can be accessed by any command that uses global array values from a compute as input. See this section for an overview of LAMMPS output options.

The array values are "intensive". The array values will be in distance^2 units.

Restrictions: none

Related commands:

compute msd

Default: none

compute pair command

Syntax:

```
compute ID group-ID pair pstyle evalue
```

- ID, group−ID are documented in compute command
- pair = style name of this compute command
- pstyle = style name of a pair style that calculates additional values
- evalue = *epair* or *evdwl* or *ecoul* or blank (optional setting)

Examples:

```
compute 1 all pair gauss
compute 1 all pair lj/cut/coul/cut ecoul
compute 1 all pair reax
```

Description:

Define a computation that extracts additional values calculated by a pair style, sums them across processors, and makes them accessible for output or further processing by other commands. The group specified for this command is ignored.

The specified *pstyle* must be a pair style used in your simulation either by itself or as a sub−style in a pair_style hybrid or hybrid/overlay command.

The *evalue* setting is optional; it may be left off the command. All pair styles tally a potential energy *epair* which may be broken into two parts: *evdwl* and *ecoul* such that *epair* = *evdwl* + *ecoul*. If the pair style calculates Coulombic interactions, their energy will be tallied in *ecoul*. Everything else (whether it is a Lennard−Jones style van der Waals interaction or not) is tallied in *evdwl*. If *evalue* is specified as *epair* or left out, then *epair* is stored as a global scalar by this compute. This is useful when using pair_style hybrid if you want to know the portion of the total energy contributed by one sub−style. If *evalue* is specfied as *evdwl* or *ecoul*, then just that portion of the energy is stored as a global scalar.

Some pair styles tally additional quantities, e.g. a breakdown of potential energy into a dozen or so components is tallied by the pair_style reax commmand. These values (1 or more) are stored as a global vector by this compute. See the doc page for individual pair styles for info on these values.

Output info:

This compute calculates a global scalar which is *epair* or *evdwl* or *ecoul*. If the pair style supports it, it also calculates a global vector of length >= 1, as determined by the pair style. These values can be used by any command that uses global scalar or vector values from a compute as input. See this section for an overview of LAMMPS output options.

The scalar and vector values calculated by this compute are "extensive".

The scalar value will be in energy units. The vector values will typically also be in energy units, but see the doc page for the pair style for details.

Restrictions: none

Related commands:

compute pe

Default:

The default for *evalue* is *epair*.

compute pair/local command

Syntax:

```
compute ID group−ID pair/local input1 input2 ...
```

- ID, group−ID are documented in compute command
- pair/local = style name of this compute command
- zero or more keywords may be appended
- keyword = *dist* or *eng* or *force*

```
  dist = tabulate pairwise distances
    eng = tablutate pairwise energies
    force = tablutate pairwise forces
```

Examples:

```
compute 1 all pair/local eng
compute 1 all pair/local dist eng force
```

Description:

Define a computation that calculates properties of individual pairwise interactions. The number of datums generated, aggregated across all processors, equals the number of pairwise interactions in the system.

The local data stored by this command is generated by looping over the pairwise neighbor list. Info about an individual pairwise interaction will only be included if both atoms in the pair are in the specified compute group, and if the current pairwise distance is less than the force cutoff distance for that interaction, as defined by the pair_style and pair_coeff commands.

The output *dist* will be in distance units. The output *eng* will be in energy units. The output *force* will be in force units.

Note that as atoms migrate from processor to processor, there will be no consistent ordering of the entries within the local vector or array from one timestep to the next. The only consistency that is guaranteed is that the ordering on a particular timestep will be the same for local vectors or arrays generated by other compute commands. For example, pair output from the compute property/local command can be combined with data from this command and output by the dump local command in a consistent way.

IMPORTANT NOTE: For pairs, if two atoms I,J are involved in 1−2, 1−3, 1−4 interactions within the molecular topology, their pairwise interaction may be turned off, and thus they may not appear in the neighbor list, and will not be part of the local data created by this command. More specifically, this may be true of I,J pairs with a weighting factor of 0.0; pairs with a non−zero weighting factor are included. The weighting factors for 1−2, 1−3, and 1−4 pairwise interactions are set by the special_bonds command.

Output info:

This compute calculates a local vector or local array depending on the number of keywords. The length of the vector or number of rows in the array is the number of pairs. If a single keyword is specified, a local vector is produced. If two or more keywords are specified, a local array is produced where the number of columns = the number of keywords. The vector or array can be accessed by any command that uses local values from a compute

as input. See this section for an overview of LAMMPS output options.

The output for *dist* will be in distance units. The output for *eng* will be in energy units. The output for *force* will be in force units.

Restrictions: none

Related commands:

dump local, compute property/local

Default: none

compute pe command

compute pe/cuda command

Syntax:

```
compute ID group−ID pe keyword ...
```

- ID, group−ID are documented in compute command
- pe = style name of this compute command
- zero or more keywords may be appended
- keyword = *pair* or *bond* or *angle* or *dihedral* or *improper* or *kspace*

Examples:

```
compute 1 all pe
compute molPE all pe bond angle dihedral improper
```

Description:

Define a computation that calculates the potential energy of the entire system of atoms. The specified group must be "all". See the compute pe/atom command if you want per−atom energies. These per−atom values could be summed for a group of atoms via the compute reduce command.

The energy is calculated by the various pair, bond, etc potentials defined for the simulation. If no extra keywords are listed, then the potential energy is the sum of pair, bond, angle, dihedral, improper, and kspace (long−range) energy. If any extra keywords are listed, then only those components are summed to compute the potential energy.

Various fixes can contribute to the total potential energy of the system. See the doc pages for individual fixes for details. The *thermo* option of the compute_modify command determines whether these contributions are added into the computed potential energy. If no keywords are specified the default is *yes*. If any keywords are specified, the default is *no*.

A compute of this style with the ID of "thermo_pe" is created when LAMMPS starts up, as if this command were in the input script:

```
compute thermo_pe all pe
```

See the "thermo_style" command for more details.

Styles with a *cuda* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in this section of the manual. The accelerated styles take the same arguments and should produce the same results, except for round−off and precision issues.

These accelerated styles are part of the "user−cuda" package. They are only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the

−suffix command−line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See this section of the manual for more instructions on how to use the accelerated styles effectively.

Output info:

This compute calculates a global scalar (the potential energy). This value can be used by any command that uses a global scalar value from a compute as input. See this section for an overview of LAMMPS output options.

The scalar value calculated by this compute is "extensive". The scalar value will be in energy units.

Restrictions: none

Related commands:

compute pe/atom

Default: none

compute pe/atom command

Syntax:

```
compute ID group−ID pe/atom keyword ...
```

- ID, group−ID are documented in compute command
- pe/atom = style name of this compute command
- zero or more keywords may be appended
- keyword = *pair* or *bond* or *angle* or *dihedral* or *improper*

Examples:

```
compute 1 all pe/atom
compute 1 all pe/atom pair
compute 1 all pe/atom pair bond
```

Description:

Define a computation that computes the per−atom potential energy for each atom in a group. See the compute pe command if you want the potential energy of the entire system.

The per−atom energy is calculated by the various pair, bond, etc potentials defined for the simulation. If no extra keywords are listed, then the potential energy is the sum of pair, bond, angle, dihedral, and improper energy. If any extra keywords are listed, then only those components are summed to compute the potential energy.

Note that the energy of each atom is due to its interaction with all other atoms in the simulation, not just with other atoms in the group.

For an energy contribution produced by a small set of atoms (e.g. 4 atoms in a dihedral or 3 atoms in a Tersoff 3−body interaction), that energy is assigned in equal portions to each atom in the set. E.g. 1/4 of the dihedral energy to each of the 4 atoms.

The dihedral_style charmm style calculates pairwise interactions between 1−4 atoms. The energy contribution of these terms is included in the pair energy, not the dihedral energy.

As an example of per−atom potential energy compared to total potential energy, these lines in an input script should yield the same result in the last 2 columns of thermo output:

```
compute         peratom all pe/atom
compute         pe all reduce sum c_peratom
thermo_style    custom step temp etotal press pe c_pe
```

IMPORTANT NOTE: The per−atom energy does NOT include contributions due to long−range Coulombic interactions (via the kspace_style command). It's not clear this contribution can easily be computed. It also does not include any Lennard−Jones tail corrections invoked by the pair_modify tail yes command, since those are global contributions to the system energy.

Output info:

This compute calculates a per−atom vector, which can be accessed by any command that uses per−atom values

from a compute as input. See this section for an overview of LAMMPS output options.

The per−atom vector values will be in energy units.

Restrictions:

Related commands:

compute pe, compute stress/atom

Default: none

compute pressure command

compute pressure/cuda command

Syntax:

```
compute ID group-ID pressure temp-ID keyword ...
```

- ID, group−ID are documented in compute command
- pressure = style name of this compute command
- temp−ID = ID of compute that calculates temperature
- zero or more keywords may be appended
- keyword = *ke* or *pair* or *bond* or *angle* or *dihedral* or *improper* or *kspace* or *fix* or *virial*

Examples:

```
compute 1 all pressure myTemp
compute 1 all pressure thermo_temp pair bond
```

Description:

Define a computation that calculates the pressure of the entire system of atoms. The specified group must be "all". See the compute stress/atom command if you want per−atom pressure (stress). These per−atom values could be summed for a group of atoms via the compute reduce command.

The pressure is computed by the formula

$$P = \frac{Nk_BT}{V} + \frac{\sum_i^N r_i \bullet f_i}{dV}$$

where N is the number of atoms in the system (see discussion of DOF below), Kb is the Boltzmann constant, T is the temperature, d is the dimensionality of the system (2 or 3 for 2d/3d), V is the system volume (or area in 2d), and the second term is the virial, computed within LAMMPS for all pairwise as well as 2−body, 3−body, and 4−body, and long−range interactions. Fixes that impose constraints (e.g. the fix shake command) also contribute to the virial term.

A symmetric pressure tensor, stored as a 6−element vector, is also calculated by this compute. The 6 components of the vector are ordered xx, yy, zz, xy, xz, yz. The equation for the I,J components (where I and J = x,y,z) is similar to the above formula, except that the first term uses components of the kinetic energy tensor and the second term uses components of the virial tensor:

$$P_{IJ} = \frac{\sum_k^N m_k v_{k_I} v_{k_J}}{V} + \frac{\sum_k^N r_{k_I} f_{k_J}}{V}$$

If no extra keywords are listed, the entire equations above are calculated which include a kinetic energy (temperature) term and the virial as the sum of pair, bond, angle, dihedral, improper, kspace (long−range), and fix contributions to the force on each atom. If any extra keywords are listed, then only those components are summed to compute temperature or ke and/or the virial. The *virial* keyword means include all terms except the kinetic

energy *ke*.

The temperature and kinetic energy tensor is not calculated by this compute, but rather by the temperature compute specified with the command. Normally this compute should calculate the temperature of all atoms for consistency with the virial term, but any compute style that calculates temperature can be used, e.g. one that excludes frozen atoms or other degrees of freedom.

Note that the N in the first formula above is really degrees−of−freedom divided by d = dimensionality, where the DOF value is calcluated by the temperature compute. See the various compute temperature styles for details.

A compute of this style with the ID of "thermo_press" is created when LAMMPS starts up, as if this command were in the input script:

```
compute thermo_press all pressure thermo_temp
```

where "thermo_temp" is the ID of a similarly defined compute of style "temp". See the "thermo_style" command for more details.

Styles with a *cuda* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in this section of the manual. The accelerated styles take the same arguments and should produce the same results, except for round−off and precision issues.

These accelerated styles are part of the "user−cuda" package. They are only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the −suffix command−line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See this section of the manual for more instructions on how to use the accelerated styles effectively.

Output info:

This compute calculates a global scalar (the pressure) and a global vector of length 6 (pressure tensor), which can be accessed by indices 1−6. These values can be used by any command that uses global scalar or vector values from a compute as input. See this section for an overview of LAMMPS output options.

The scalar and vector values calculated by this compute are "intensive". The scalar and vector values will be in pressure units.

Restrictions: none

Related commands:

compute temp, compute stress/atom, thermo_style,

Default: none

compute property/atom command

Syntax:

```
compute ID group-ID property/atom input1 input2 ...
```

- ID, group−ID are documented in compute command
- property/atom = style name of this compute command
- input = one or more atom attributes

```
possible attributes = id, mol, type, mass,
                      x, y, z, xs, ys, zs, xu, yu, zu, ix, iy, iz,
                      vx, vy, vz, fx, fy, fz,
                      q, mux, muy, muz, mu,
                      radius, diameter, omegax, omegay, omegaz,
                      angmomx, angmomy, angmomz,
                      shapex,shapey, shapez,
                      quatw, quati, quatj, quatk, tqx, tqy, tqz,
                      spin, eradius, ervel, erforce

  id = atom ID
  mol = molecule ID
  type = atom type
  mass = atom mass
  x,y,z = unscaled atom coordinates
  xs,ys,zs = scaled atom coordinates
  xu,yu,zu = unwrapped atom coordinates
  ix,iy,iz = box image that the atom is in
  vx,vy,vz = atom velocities
  fx,fy,fz = forces on atoms
  q = atom charge
  mux,muy,muz = orientation of dipole moment of atom
  mu = magnitude of dipole moment of atom
  radius,diameter = radius,diameter of spherical particle
  omegax,omegay,omegaz = angular velocity of extended particle
  angmomx,angmomy,angmomz = angular momentum of extended particle
  tqx,tqy,tqz = torque on extended particles
  spin = electron spin
  eradius = electron radius
  ervel = electron radial velocity
  erforce = electron radial force
  shapex,shapey,shapez = 3 diameters of aspherical particle
  quatw,quati,quatj,quatk = quaternion components for aspherical particles
```

Examples:

```
compute 1 all property/atom xs vx fx mux
compute 2 all property/atom type
compute 1 all property/atom ix iy iz
```

Description:

Define a computation that simply stores atom attributes for each atom in the group. This is useful so that the values can be used by other output commands that take computes as inputs. See for example, the compute reduce, fix ave/atom, fix ave/histo, fix ave/spatial, and atom−style variable commands.

The list of possible attributes is the same as that used by the dump custom command, which describes their meaning, with some additional quantities that are only defined for certain atom styles. Basically, this list gives your input script access to any per−atom quantity stored by LAMMPS.

The values are stored in a per−atom vector or array as discussed below. Zeroes are stored for atoms not in the specified group.

The additional quantities only accessible via this command (and not directly via the dump custom command are as follows.

Shapex, *shapey*, and *shapez* are defined for ellipsoidal particles and define the 3d shape of each particle. *Quatw*, *quati*, *quatj*, and *quatk* are also defined for ellipsoidal particles and store the 4−vector quaternion representing the orientation of each particle. See the set command for an explanation of the quaternion vector.

Output info:

This compute calculates a per−atom vector or per−atom array depending on the number of input values. If a single input is specified, a per−atom vector is produced. If two or more inputs are specified, a per−atom array is produced where the number of columns = the number of inputs. The vector or array can be accessed by any command that uses per−atom values from a compute as input. See this section for an overview of LAMMPS output options.

The vector or array values will be in whatever units the corresponding attribute is in, e.g. velocity units for vx, charge units for q, etc.

Restrictions: none

Related commands:

dump custom, compute reduce, fix ave/atom, fix ave/spatial

Default: none

compute property/local command

Syntax:

```
compute ID group-ID property/local input1 input2 ...
```

- ID, group−ID are documented in compute command
- property/local = style name of this compute command
- input = one or more attributes

```
possible attributes = natom1 natom2
                      patom1 patom2
                      batom1 batom2 btype
                      aatom1 aatom2 aatom3 atype
                      datom1 datom2 datom3 dtype
                      iatom1 iatom2 iatom3 itype

  natom1, natom2 = IDs of 2 atoms in each pair (within neighbor cutoff)
  patom1, patom2 = IDs of 2 atoms in each pair (within force cutoff)
  batom1, batom2 = IDs of 2 atoms in each bond
  btype = bond type of each bond
  aatom1, aatom2, aatom3 = IDs of 3 atoms in each angle
  atype = angle type of each angle
  datom1, datom2, datom3, datom4 = IDs of 4 atoms in each dihedral
  dtype = dihedral type of each dihedral
  iatom1, iatom2, iatom3, iatom4 = IDs of 4 atoms in each improper
  itype = improper type of each improper
```

Examples:

```
compute 1 all property/local btype batom1 batom2
compute 1 all property/local atype aatom2
```

Description:

Define a computation that stores the specified attributes as local data so it can be accessed by other output commands. If the input attributes refer to bond information, then the number of datums generated, aggregated across all processors, equals the number of bonds in the system. Ditto for pairs, angles, etc.

If multiple input attributes are specified then they must all generate the same amount of information, so that the resulting local array has the same number of rows for each column. This means that only bond attributes can be specified together, or angle attributes, etc. Bond and angle attributes can not be mixed in the same compute property/local command.

If the inputs are pair attributes, the local data is generated by looping over the pairwise neighbor list. Info about an individual pairwise interaction will only be included if both atoms in the pair are in the specified compute group. For *natom1* and *natom2*, all atom pairs in the neighbor list are considered (out to the neighbor cutoff = force cutoff + neighbor skin). For *patom1* and *patom2*, the distance between the atoms must be less than the force cutoff distance for that pair to be included, as defined by the pair_style and pair_coeff commands.

If the inputs are bond, angle, etc attributes, the local data is generated by looping over all the atoms owned on a processor and extracting bond, angle, etc info. For bonds, info about an individual bond will only be included if both atoms in the bond are in the specified compute group. Likewise for angles, dihedrals, etc.

Note that as atoms migrate from processor to processor, there will be no consistent ordering of the entries within the local vector or array from one timestep to the next. The only consistency that is guaranteed is that the ordering on a particular timestep will be the same for local vectors or arrays generated by other compute commands. For example, output from the compute bond/local command can be combined with bond atom indices from this command and output by the dump local command in a consistent way.

The *natom1* and *natom2*, or *patom1* and *patom2* attributes refer to the atom IDs of the 2 atoms in each pairwise interaction computed by the pair_style command.

IMPORTANT NOTE: For pairs, if two atoms I,J are involved in 1−2, 1−3, 1−4 interactions within the molecular topology, their pairwise interaction may be turned off, and thus they may not appear in the neighbor list, and will not be part of the local data created by this command. More specifically, this may be true of I,J pairs with a weighting factor of 0.0; pairs with a non−zero weighting factor are included. The weighting factors for 1−2, 1−3, and 1−4 pairwise interactions are set by the special_bonds command.

The *batom1* and *batom2* attributes refer to the atom IDs of the 2 atoms in each bond. The *btype* attribute refers to the type of the bond, from 1 to Nbtypes = # of bond types. The number of bond types is defined in the data file read by the read_data command. The attributes that start with "a", "d", "i", refer to similar values for angles, dihedrals, and impropers.

Output info:

This compute calculates a local vector or local array depending on the number of input values. The length of the vector or number of rows in the array is the number of bonds, angles, etc. If a single input is specified, a local vector is produced. If two or more inputs are specified, a local array is produced where the number of columns = the number of inputs. The vector or array can be accessed by any command that uses local values from a compute as input. See this section for an overview of LAMMPS output options.

The vector or array values will be integers that correspond to the specified attribute.

Restrictions: none

Related commands:

dump local, compute reduce

Default: none

compute property/molecule command

Syntax:

```
compute ID group−ID property/molecule input1 input2 ...
```

- ID, group−ID are documented in compute command
- property/molecule = style name of this compute command
- input = one or more attributes

```
possible attributes = mol cout
  mol = molecule ID
  count = # of atoms in molecule
```

Examples:

```
compute 1 all property/molecule mol
```

Description:

Define a computation that stores the specified attributes as global data so it can be accessed by other output commands and used in conjunction with other commands that generate per−molecule data, such as compute com/molecule and compute msd/molecule.

The ordering of per−molecule quantities produced by this compute is consistent with the ordering produced by other compute commands that generate per−molecule datums. Conceptually, them molecule IDs will be in ascending order for any molecule with one or more of its atoms in the specified group.

The *mol* attribute is the molecule ID. This attribute can be used to produce molecule IDs as labels for per−molecule datums generated by other computes or fixes when they are output to a file, e.g. by the fix ave/time command.

The *count* attribute is the number of atoms in the molecule.

Output info:

This compute calculates a global vector or global array depending on the number of input values. The length of the vector or number of rows in the array is the number of molecules. If a single input is specified, a global vector is produced. If two or more inputs are specified, a global array is produced where the number of columns = the number of inputs. The vector or array can be accessed by any command that uses global values from a compute as input. See this section for an overview of LAMMPS output options.

The vector or array values will be integers that correspond to the specified attribute.

Restrictions: none

Related commands: none

Default: none

compute rdf command

Syntax:

```
compute ID group-ID rdf Nbin itype1 jtype1 itype2 jtype2 ...
```

- ID, group−ID are documented in compute command
- rdf = style name of this compute command
- Nbin = number of RDF bins
- itypeN = central atom type for Nth RDF histogram (see asterisk form below)
- jtypeN = distribution atom type for Nth RDF histogram (see asterisk form below)

Examples:

```
compute 1 all rdf 100
compute 1 all rdf 100 1 1
compute 1 all rdf 100 * 3
compute 1 fluid rdf 500 1 1 1 2 2 1 2 2
compute 1 fluid rdf 500 1*3 2 5 *10
```

Description:

Define a computation that calculates the radial distribution function (RDF), also called g(r), and the coordination number for a group of particles. Both are calculated in histogram form by binning pairwise distances into *Nbin* bins from 0.0 to the maximum force cutoff defined by the pair_style command. The bins are of uniform size in radial distance. Thus a single bin encompasses a thin shell of distances in 3d and a thin ring of distances in 2d.

The *itypeN* and *jtypeN* arguments are optional. These arguments must come in pairs. If no pairs are listed, then a single histogram is computed for g(r) between all atom types. If one or more pairs are listed, then a separate histogram is generated for each *itype*,*jtype* pair.

The *itypeN* and *jtypeN* settings can be specified in one of two ways. An explicit numeric value can be used, as in the 4th example above. Or a wild−card asterisk can be used to specify a range of atom types. This takes the form "*" or "*n" or "n*" or "m*n". If N = the number of atom types, then an asterisk with no numeric values means all types from 1 to N. A leading asterisk means all types from 1 to n (inclusive). A trailing asterisk means all types from n to N (inclusive). A middle asterisk means all types from m to n (inclusive).

If both *itypeN* and *jtypeN* are single values, as in the 4th example above, this means that a g(r) is computed where atoms of type *itypeN* are the central atom, and atoms of type *jtypeN* are the distribution atom. If either *itypeN* and *jtypeN* represent a range of values via the wild−card asterisk, as in the 5th example above, this means that a g(r) is computed where atoms of any of the range of types represented by *itypeN* are the central atom, and atoms of any of the range of types represented by *jtypeN* are the distribution atom.

Pairwise distances are generated by looping over a pairwise neighbor list, just as they would be in a pair_style computation. The distance between two atoms I and J is included in a specific histogram if the following criteria are met:

- atoms I,J are both in the specified compute group
- the distance between atoms I,J is less than the maximum force cutoff
- the type of the I atom matches itypeN (one or a range of types)
- the type of the J atom matches jtypeN (one or a range of types)

It is OK if a particular pairwise distance is included in more than one individual histogram, due to the way the *itypeN* and *jtypeN* arguments are specified.

The g(r) value for a bin is calculated from the histogram count by scaling it by the idealized number of how many counts there would be if atoms of type *jtypeN* were uniformly distributed. Thus it involves the count of *itypeN* atoms, the count of *jtypeN* atoms, the volume of the entire simulation box, and the volume of the bin's thin shell in 3d (or the area of the bin's thin ring in 2d).

A coordination number coord(r) is also calculated, which is the sum of g(r) values for all bins up to and including the current bin.

The simplest way to output the results of the compute rdf calculation to a file is to use the fix ave/time command, for example:

```
compute myRDF all rdf 50
fix 1 all ave/time 100 1 100 c_myRDF file tmp.rdf mode vector
```

Output info:

This compute calculates a global array with the number of rows = *Nbins*, and the number of columns = 1 + 2*Npairs, where Npairs is the number of I,J pairings specified. The first column has the bin coordinate (center of the bin), Each successive set of 2 columns has the g(r) and coord(r) values for a specific set of *itypeN* versus *jtypeN* interactions, as described above. These values can be used by any command that uses a global values from a compute as input. See this section for an overview of LAMMPS output options.

The array values calculated by this compute are all "intensive".

The first column of array values will be in distance units. The g(r) columns of array values are normalized numbers >= 0.0. The coordination number columns of array values are also numbers >= 0.0.

Restrictions:

The RDF is not computed for distances longer than the force cutoff, since processors (in parallel) don't know about atom coordinates for atoms further away than that distance. If you want an RDF for larger distances, you'll need to post−process a dump file.

Related commands:

fix ave/time

Default: none

compute reduce command

compute reduce/region command

Syntax:

```
compute ID group-ID style arg mode input1 input2 ... keyword args ...
```

- ID, group−ID are documented in compute command
- style = *reduce* or *reduce/region*

```
  reduce arg = none
    reduce/region arg = region-ID
      region-ID = ID of region to use for choosing atoms
```

- mode = *sum* or *min* or *max* or *ave*
- one or more inputs can be listed
- input = x, y, z, vx, vy, vz, fx, fy, fz, c_ID, c_ID[N], f_ID, f_ID[N], v_name

```
    x,y,z,vx,vy,vz,fx,fy,fz = atom attribute (position, velocity, force component)
    c_ID = per-atom or local vector calculated by a compute with ID
    c_ID[I] = Ith column of per-atom or local array calculated by a compute with ID
    f_ID = per-atom or local vector calculated by a fix with ID
    f_ID[I] = Ith column of per-atom or local array calculated by a fix with ID
    v_name = per-atom vector calculated by an atom-style variable with name
```

- zero or more keyword/args pairs may be appended
- keyword = *replace*

```
  replace args = vec1 vec2
      vec1 = reduced value from this input vector will be replaced
      vec2 = replace it with vec1[N] where N is index of max/min value from vec2
```

Examples:

```
compute 1 all reduce sum c_force
compute 1 all reduce/region subbox sum c_force
compute 2 all reduce min c_press2 f_ave v_myKE
compute 3 fluid reduce max c_index1 c_index2 c_dist replace 1 3 replace 2 3
```

Description:

Define a calculation that "reduces" one or more vector inputs into scalar values, one per listed input. The inputs can be per−atom or local quantities; they cannot be global quantities. Atom attributes are per−atom quantities, computes and fixes may generate any of the three kinds of quantities, and atom−style variables generate per−atom quantities. See the variable command and its special functions which can perform the same operations as the compute reduce command on global vectors.

The reduction operation is specified by the *mode* setting. The *sum* option adds the values in the vector into a global total. The *min* or *max* options find the minimum or maximum value across all vector values. The *ave* setting adds the vector values into a global total, then divides by the number of values in the vector.

Each listed input is operated on independently. For per−atom inputs, the group specified with this command means only atoms within the group contribute to the result. For per−atom inputs, if the compute reduce/region command is used, the atoms must also currently be within the region. Note that an input that produces per−atom

quantities may define its own group which affects the quantities it returns. For example, if a compute is used as an input which generates a per−atom vector, it will generate values of 0.0 for atoms that are not in the group specified for that compute.

Each listed input can be an atom attribute (position, velocity, force component) or can be the result of a compute or fix or the evaluation of an atom−style variable.

The atom attribute values (x,y,z,vx,vy,vz,fx,fy,fz) are self−explanatory. Note that other atom attributes can be used as inputs to this fix by using the compute property/atom command and then specifying an input value from that compute.

If a value begins with "c_", a compute ID must follow which has been previously defined in the input script. Computes can generate per−atom or local quantities. See the individual compute doc page for details. If no bracketed integer is appended, the vector calculated by the compute is used. If a bracketed interger is appended, the Ith column of the array calculated by the compute is used. Users can also write code for their own compute styles and add them to LAMMPS.

If a value begins with "f_", a fix ID must follow which has been previously defined in the input script. Fixes can generate per−atom or local quantities. See the individual fix doc page for details. Note that some fixes only produce their values on certain timesteps, which must be compatible with when compute reduce references the values, else an error results. If no bracketed integer is appended, the vector calculated by the fix is used. If a bracketed integer is appended, the Ith column of the array calculated by the fix is used. Users can also write code for their own fix style and add them to LAMMPS.

If a value begins with "v_", a variable name must follow which has been previously defined in the input script. It must be an atom−style variable. Atom−style variables can reference thermodynamic keywords and various per−atom attributes, or invoke other computes, fixes, or variables when they are evaluated, so this is a very general means of generating per−atom quantities to reduce.

If the *replace* keyword is used, two indices *vec1* and *vec2* are specified, where each index ranges from 1 to the # of input values. The replace keyword can only be used if the *mode* is *min* or *max*. It works as follows. A min/max is computed as usual on the *vec2* input vector. The index N of that value within *vec2* is also stored. Then, instead of performing a min/max on the *vec1* input vector, the stored index is used to select the Nth element of the *vec1* vector.

Thus, for example, if you wish to use this compute to find the bond with maximum stretch, you can do it as follows:

```
compute 1 all property/local batom1 batom2
compute 2 all bond/local dist
compute 3 all reduce max c_1[1] c_1[2] c_2 replace 1 3 replace 2 3
thermo_style custom step temp c_3[1] c_3[2] c_3[3]
```

The first two input values in the compute reduce command are vectors with the IDs of the 2 atoms in each bond, using the compute property/local command. The last input value is bond distance, using the compute bond/local command. Instead of taking the max of the two atom ID vectors, which does not yield useful information in this context, the *replace* keywords will extract the atom IDs for the two atoms in the bond of maximum stretch. These atom IDs and the bond stretch will be printed with thermodynamic output.

If a single input is specified this compute produces a global scalar value. If multiple inputs are specified, this compute produces a global vector of values, the length of which is equal to the number of inputs specified.

As discussed below, for *sum* mode, the value(s) produced by this compute are all "extensive", meaning their value scales linearly with the number of atoms involved. If normalized values are desired, this compute can be accessed by the thermo_style custom command with thermo_modify norm yes set as an option. Or it can be accessed by a variable that divides by the appropriate atom count.

Output info:

This compute calculates a global scalar if a single input value is specified or a global vector of length N where N is the number of inputs, and which can be accessed by indices 1 to N. These values can be used by any command that uses global scalar or vector values from a compute as input. See this section for an overview of LAMMPS output options.

All the scalar or vector values calculated by this compute are "intensive", except when the *sum* mode is used on per−atom or local vectors, in which case the calculated values are "extensive".

The scalar or vector values will be in whatever units the quantities being reduced are in.

Restrictions: none

Related commands:

compute, fix, variable

Default: none

compute slice command

Syntax:

```
compute ID group-ID slice Nstart Nstop Nskip input1 input2 ...
```

- ID, group−ID are documented in compute command
- slice = style name of this compute command
- Nstart = starting index within input vector(s)
- Nstop = stopping index within input vector(s)
- Nskip = extract every Nskip elements from input vector(s)
- input = c_ID, c_ID[N], f_ID, f_ID[N]

```
c_ID = global vector calculated by a compute with ID
c_ID[I] = Ith column of global array calculated by a compute with ID
f_ID = global vector calculated by a fix with ID
f_ID[I] = Ith column of global array calculated by a fix with ID
```

Examples:

```
compute 1 all slice 1 100 10 c_msdmol[4]
compute 1 all slice 301 400 1 c_msdmol[4]
```

Description:

Define a calculation that "slices" one or more vector inputs into smaller vectors, one per listed input. The inputs can be global quantities; they cannot be per−atom or local quantities. Computes and fixes may generate any of the three kinds of quantities. Variables do not generate global vectors. The group specified with this command is ignored.

The values extracted from the input vector(s) are determined by the *Nstart*, *Nstop*, and *Nskip* parameters. The elements of an input vector of length N are indexed from 1 to N. Starting at element *Nstart*, every Mth element is extracted, where M = *Nskip*, until element *Nstop* is reached. The extracted quantities are stored as a vector, which is typically shorter than the input vector.

Each listed input is operated on independently to produce one output vector. Each listed input must be a global vector or column of a global array calculated by another compute or fix.

If an input value begins with "c_", a compute ID must follow which has been previously defined in the input script and which generates a global vector or array. See the individual compute doc page for details. If no bracketed integer is appended, the vector calculated by the compute is used. If a bracketed interger is appended, the Ith column of the array calculated by the compute is used. Users can also write code for their own compute styles and add them to LAMMPS.

If a value begins with "f_", a fix ID must follow which has been previously defined in the input script and which generates a global vector or array. See the individual fix doc page for details. Note that some fixes only produce their values on certain timesteps, which must be compatible with when compute slice references the values, else an error results. If no bracketed integer is appended, the vector calculated by the fix is used. If a bracketed integer is appended, the Ith column of the array calculated by the fix is used. Users can also write code for their own fix style and add them to LAMMPS.

If a single input is specified this compute produces a global vector, even if the length of the vector is 1. If multiple inputs are specified, then a global array of values is produced, with the number of columns equal to the number of inputs specified.

Output info:

This compute calculates a global vector if a single input value is specified or a global array with N columns where N is the number of inputs. The length of the vector or the number of rows in the array is equal to the number of values extracted from each input vector. These values can be used by any command that uses global vector or array values from a compute as input. See this section for an overview of LAMMPS output options.

The vector or array values calculated by this compute are simply copies of values generated by computes or fixes that are input vectors to this compute. If there is a single input vector of intensive and/or extensive values, then each value in the vector of values calculated by this compute will be "intensive" or "extensive", depending on the corresponding input value. If there are multiple input vectors, and all the values in them are intensive, then the array values calculated by this compute are "intensive". If there are multiple input vectors, and any value in them is extensive, then the array values calculated by this compute are "extensive".

The vector or array values will be in whatever units the input quantities are in.

Restrictions: none

Related commands:

compute, fix, compute reduce

Default: none

compute stress/atom command

Syntax:

```
compute ID group-ID stress/atom keyword ...
```

- ID, group−ID are documented in compute command
- stress/atom = style name of this compute command
- zero or more keywords may be appended
- keyword = *ke* or *pair* or *bond* or *angle* or *dihedral* or *improper* or *fix* or *virial*

Examples:

```
compute 1 mobile stress/atom
compute 1 all stress/atom pair bond
```

Description:

Define a computation that computes the symmetric per−atom stress tensor for each atom in a group. The tensor for each atom has 6 components and is stored as a 6−element vector in the following order: xx, yy, zz, xy, xz, yz. See the compute pressure command if you want the stress tensor (pressure) of the entire system.

The stress tensor for atom *I* is given by the following formula, where *a* and *b* take on values x,y,z to generate the 6 components of the symmetric tensor:

$$
\begin{aligned}
S_{ab} = -\Bigg[& m v_a v_b + \frac{1}{2}\sum_{n=1}^{N_p}(r_{1_a}F_{1_b} + r_{2_a}F_{2_b}) + \frac{1}{2}\sum_{n=1}^{N_b}(r_{1_a}F_{1_b} + r_{2_a}F_{2_b}) + \\
& \frac{1}{3}\sum_{n=1}^{N_a}(r_{1_a}F_{1_b} + r_{2_a}F_{2_b} + r_{3_a}F_{3_b}) + \frac{1}{4}\sum_{n=1}^{N_d}(r_{1_a}F_{1_b} + r_{2_a}F_{2_b} + r_{3_a}F_{3_b} + r_{4_a}F_{4_b}) + \\
& \frac{1}{4}\sum_{n=1}^{N_i}(r_{1_a}F_{1_b} + r_{2_a}F_{2_b} + r_{3_a}F_{3_b} + r_{4_a}F_{4_b}) + \sum_{n=1}^{N_f} r_{i_a}F_{i_b} \Bigg]
\end{aligned}
$$

The first term is a kinetic energy contribution for atom *I*. The second term is a pairwise energy contribution where *n* loops over the *Np* neighbors of atom *I*, *r1* and *r2* are the positions of the 2 atoms in the pairwise interaction, and *F1* and *F2* are the forces on the 2 atoms resulting from the pairwise interaction. The third term is a bond contribution of similar form for the *Nb* bonds which atom *I* is part of. There are similar terms for the *Na* angle, *Nd* dihedral, and *Ni* improper interactions atom *I* is part of. Finally, there is a term for the *Nf* fixes that apply internal constraint forces to atom *I*. Currently, only the fix shake and fix rigid commands contribute to this term.

As the coefficients in the formula imply, a virial contribution produced by a small set of atoms (e.g. 4 atoms in a dihedral or 3 atoms in a Tersoff 3−body interaction) is assigned in equal portions to each atom in the set. E.g. 1/4 of the dihedral virial to each of the 4 atoms, or 1/3 of the fix virial due to SHAKE constraints applied to atoms in a a water molecule via the fix shake command.

If no extra keywords are listed, all of the terms in this formula are included in the per−atom stress tensor. If any extra keywords are listed, only those terms are summed to compute the tensor. The *virial* keyword means include

all terms except the kinetic energy *ke*.

Note that the stress for each atom is due to its interaction with all other atoms in the simulation, not just with other atoms in the group.

The dihedral_style charmm style calculates pairwise interactions between 1−4 atoms. The virial contribution of these terms is included in the pair virial, not the dihedral virial.

Note that as defined in the formula, per−atom stress is the negative of the per−atom pressure tensor. It is also really a stress*volume formulation, meaning the computed quantity is in units of pressure*volume. It would need to be divided by a per−atom volume to have units of stress (pressure), but an individual atom's volume is not well defined or easy to compute in a deformed solid or a liquid. Thus, if the diagonal components of the per−atom stress tensor are summed for all atoms in the system and the sum is divided by dV, where d = dimension and V is the volume of the system, the result should be −P, where P is the total pressure of the system.

These lines in an input script for a 3d system should yield that result. I.e. the last 2 columns of thermo output will be the same:

```
compute         peratom all stress/atom
compute         p all reduce sum c_peratom[1] c_peratom[2] c_peratom[3]
variable        press equal -(c_p[1]+c_p[2]+c_p[3])/(3*vol)
thermo_style    custom step temp etotal press v_press
```

IMPORTANT NOTE: The per−atom stress does NOT include contributions due to long−range Coulombic interactions (via the kspace_style command). It's not clear this contribution can easily be computed.

Output info:

This compute calculates a per−atom array with 6 columns, which can be accessed by indices 1−6 by any command that uses per−atom values from a compute as input. See this section for an overview of LAMMPS output options.

The per−atom array values will be in pressure*volume units as discussed above.

Restrictions: none

Related commands:

compute pe, compute pressure

Default: none

compute temp command

compute temp/cuda command

Syntax:

```
compute ID group-ID temp
```

- ID, group−ID are documented in compute command
- temp = style name of this compute command

Examples:

```
compute 1 all temp
compute myTemp mobile temp
```

Description:

Define a computation that calculates the temperature of a group of atoms. A compute of this style can be used by any command that computes a temperature, e.g. thermo_modify, fix temp/rescale, fix npt, etc.

The temperature is calculated by the formula KE = dim/2 N k T, where KE = total kinetic energy of the group of atoms (sum of 1/2 m v^2), dim = 2 or 3 = dimensionality of the simulation, N = number of atoms in the group, k = Boltzmann constant, and T = temperature.

A kinetic energy tensor, stored as a 6−element vector, is also calculated by this compute for use in the computation of a pressure tensor. The formula for the components of the tensor is the same as the above formula, except that v^2 is replaced by vx*vy for the xy component, etc. The 6 components of the vector are ordered xx, yy, zz, xy, xz, yz.

The number of atoms contributing to the temperature is assumed to be constant for the duration of the run; use the *dynamic* option of the compute_modify command if this is not the case.

This compute subtracts out degrees−of−freedom due to fixes that constrain molecular motion, such as fix shake and fix rigid. This means the temperature of groups of atoms that include these constraints will be computed correctly. If needed, the subtracted degrees−of−freedom can be altered using the *extra* option of the compute_modify command.

A compute of this style with the ID of "thermo_temp" is created when LAMMPS starts up, as if this command were in the input script:

```
compute thermo_temp all temp
```

See the "thermo_style" command for more details.

See this howto section of the manual for a discussion of different ways to compute temperature and perform thermostatting.

Styles with a *cuda* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in this section of the manual. The

accelerated styles take the same arguments and should produce the same results, except for round−off and precision issues.

These accelerated styles are part of the "user−cuda" package. They are only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the −suffix command−line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See this section of the manual for more instructions on how to use the accelerated styles effectively.

Output info:

This compute calculates a global scalar (the temperature) and a global vector of length 6 (KE tensor), which can be accessed by indices 1−6. These values can be used by any command that uses global scalar or vector values from a compute as input. See this section for an overview of LAMMPS output options.

The scalar value calculated by this compute is "intensive". The vector are "extensive".

The scalar value will be in temperature units. The vector values will be in energy units.

Restrictions: none

Related commands:

compute temp/partial, compute temp/region, compute pressure

Default: none

compute temp/asphere command

Syntax:

```
compute ID group-ID temp/asphere keyword value ...
```

- ID, group−ID are documented in compute command
- temp/asphere = style name of this compute command
- zero or more keyword/value pairs may be appended
- keyword = *bias* or *dof*

```
  bias value = bias-IDuniform or gaussian
      bias-ID = ID of a temperature compute that removes a velocity bias
    dof value = all or rotate
      all = compute temperature of translational and rotational degrees of freedom
      rotate = compute temperature of just rotational degrees of freedom
```

Examples:

```
compute 1 all temp/asphere
compute myTemp mobile temp/asphere bias tempCOM
compute myTemp mobile temp/asphere dof rotate
```

Description:

Define a computation that calculates the temperature of a group of aspherical particles, including a contribution from both their translational and rotational kinetic energy. This differs from the usual compute temp command, which assumes point particles with only translational kinetic energy.

Only finite−size particles (aspherical or spherical) can be included in the group. For 3d finite−size particles, each has 6 degrees of freedom (3 translational, 3 rotational). For 2d finite−size particles, each has 3 degrees of freedom (2 translational, 1 rotational).

IMPORTANT NOTE: This choice for degrees of freedom (dof) assumes that all finite−size aspherical or spherical particles in your model will freely rotate, sampling all their rotational dof. It is possible to use a combination of interaction potentials and fixes that induce no torque or otherwise constrain some of all of your particles so that this is not the case. Then there are less dof and you should use the compute_modify extra command to adjust the dof accordingly.

For example, an aspherical particle with all three of its shape parameters the same is a sphere. If it does not rotate, then it should have 3 dof instead of 6 in 3d (or 2 instead of 3 in 2d). A uniaxial aspherical particle has two of its three shape parameters the same. If it does not rotate around the axis perpendicular to its circular cross section, then it should have 5 dof instead of 6 in 3d.

The translational kinetic energy is computed the same as is described by the compute temp command. The rotational kinetic energy is computed as 1/2 I w^2, where I is the inertia tensor for the aspherical particle and w is its angular velocity, which is computed from its angular momentum.

IMPORTANT NOTE: For 2d models, particles are treated as ellipsoids, not ellipses, meaning their moments of inertia will be the same as in 3d.

A kinetic energy tensor, stored as a 6−element vector, is also calculated by this compute. The formula for the components of the tensor is the same as the above formula, except that v^2 and w^2 are replaced by vx*vy and wx*wy for the xy component, and the appropriate elements of the inertia tensor are used. The 6 components of the vector are ordered xx, yy, zz, xy, xz, yz.

The number of atoms contributing to the temperature is assumed to be constant for the duration of the run; use the *dynamic* option of the compute_modify command if this is not the case.

This compute subtracts out translational degrees−of−freedom due to fixes that constrain molecular motion, such as fix shake and fix rigid. This means the temperature of groups of atoms that include these constraints will be computed correctly. If needed, the subtracted degrees−of−freedom can be altered using the *extra* option of the compute_modify command.

See this howto section of the manual for a discussion of different ways to compute temperature and perform thermostatting.

The keyword/value option pairs are used in the following ways.

For the *bias* keyword, *bias−ID* refers to the ID of a temperature compute that removes a "bias" velocity from each atom. This allows compute temp/sphere to compute its thermal temperature after the translational kinetic energy components have been altered in a prescribed way, e.g. to remove a velocity profile. Thermostats that use this compute will work with this bias term. See the doc pages for individual computes that calculate a temperature and the doc pages for fixes that perform thermostatting for more details.

For the *dof* keyword, a setting of *all* calculates a temperature that includes both translational and rotational degrees of freedom. A setting of *rotate* calculates a temperature that includes only rotational degrees of freedom.

Output info:

This compute calculates a global scalar (the temperature) and a global vector of length 6 (KE tensor), which can be accessed by indices 1−6. These values can be used by any command that uses global scalar or vector values from a compute as input. See this section for an overview of LAMMPS output options.

The scalar value calculated by this compute is "intensive". The vector values are "extensive".

The scalar value will be in temperature units. The vector values will be in energy units.

Restrictions:

This compute is part of the "asphere" package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

This compute requires that atoms store angular momementum and a quaternion as defined by the atom_style ellipsoid command.

All particles in the group must be finite−size. They cannot be point particles, but they can be aspherical or spherical as defined by their shape attribute.

Related commands:

compute temp

Default: none

compute temp/com command

Syntax:

```
compute ID group-ID temp/com
```

- ID, group−ID are documented in compute command
- temp/com = style name of this compute command

Examples:

```
compute 1 all temp/com
compute myTemp mobile temp/com
```

Description:

Define a computation that calculates the temperature of a group of atoms, after subtracting out the center−of−mass velocity of the group. This is useful if the group is expected to have a non−zero net velocity for some reason. A compute of this style can be used by any command that computes a temperature, e.g. thermo_modify, fix temp/rescale, fix npt, etc.

After the center−of−mass velocity has been subtracted from each atom, the temperature is calculated by the formula KE = dim/2 N k T, where KE = total kinetic energy of the group of atoms (sum of 1/2 m v^2), dim = 2 or 3 = dimensionality of the simulation, N = number of atoms in the group, k = Boltzmann constant, and T = temperature.

A kinetic energy tensor, stored as a 6−element vector, is also calculated by this compute for use in the computation of a pressure tensor. The formula for the components of the tensor is the same as the above formula, except that v^2 is replaced by vx*vy for the xy component, etc. The 6 components of the vector are ordered xx, yy, zz, xy, xz, yz.

The number of atoms contributing to the temperature is assumed to be constant for the duration of the run; use the *dynamic* option of the compute_modify command if this is not the case.

The removal of the center−of−mass velocity by this fix is essentially computing the temperature after a "bias" has been removed from the velocity of the atoms. If this compute is used with a fix command that performs thermostatting then this bias will be subtracted from each atom, thermostatting of the remaining thermal velocity will be performed, and the bias will be added back in. Thermostatting fixes that work in this way include fix nvt, fix temp/rescale, fix temp/berendsen, and fix langevin.

This compute subtracts out degrees−of−freedom due to fixes that constrain molecular motion, such as fix shake and fix rigid. This means the temperature of groups of atoms that include these constraints will be computed correctly. If needed, the subtracted degrees−of−freedom can be altered using the *extra* option of the compute_modify command.

See this howto section of the manual for a discussion of different ways to compute temperature and perform thermostatting.

Output info:

This compute calculates a global scalar (the temperature) and a global vector of length 6 (KE tensor), which can be accessed by indices 1−6. These values can be used by any command that uses global scalar or vector values from a compute as input. See this section for an overview of LAMMPS output options.

The scalar value calculated by this compute is "intensive". The vector values are "extensive".

The scalar value will be in temperature units. The vector values will be in energy units.

Restrictions: none

Related commands:

compute temp

Default: none

compute temp/deform command

Syntax:

```
compute ID group-ID temp/deform
```

- ID, group−ID are documented in compute command
- temp/deform = style name of this compute command

Examples:

```
compute myTemp all temp/deform
```

Description:

Define a computation that calculates the temperature of a group of atoms, after subtracting out a streaming velocity induced by the simulation box changing size and/or shape, for example in a non−equilibrium MD (NEMD) simulation. The size/shape change is induced by use of the fix deform command. A compute of this style is created by the fix nvt/sllod command to compute the thermal temperature of atoms for thermostatting purposes. A compute of this style can also be used by any command that computes a temperature, e.g. thermo_modify, fix temp/rescale, fix npt, etc.

The deformation fix changes the box size and/or shape over time, so each atom in the simulation box can be thought of as having a "streaming" velocity. For example, if the box is being sheared in x, relative to y, then atoms at the bottom of the box (low y) have a small x velocity, while atoms at the top of the box (hi y) have a large x velocity. This position−dependent streaming velocity is subtracted from each atom's actual velocity to yield a thermal velocity which is used to compute the temperature.

IMPORTANT NOTE: Fix deform has an option for remapping either atom coordinates or velocities to the changing simulation box. When using this compute in conjunction with a deforming box, fix deform should NOT remap atom positions, but rather should let atoms respond to the changing box by adjusting their own velocities (or let fix deform remap the atom velocities, see it's remap option). If fix deform does remap atom positions, then they appear to move with the box but their velocity is not changed, and thus they do NOT have the streaming velocity assumed by this compute. LAMMPS will warn you if fix deform is defined and its remap setting is not consistent with this compute.

After the streaming velocity has been subtracted from each atom, the temperature is calculated by the formula KE = dim/2 N k T, where KE = total kinetic energy of the group of atoms (sum of 1/2 m v^2), dim = 2 or 3 = dimensionality of the simulation, N = number of atoms in the group, k = Boltzmann constant, and T = temperature. Note that v in the kinetic energy formula is the atom's thermal velocity.

A kinetic energy tensor, stored as a 6−element vector, is also calculated by this compute for use in the computation of a pressure tensor. The formula for the components of the tensor is the same as the above formula, except that v^2 is replaced by vx*vy for the xy component, etc. The 6 components of the vector are ordered xx, yy, zz, xy, xz, yz.

The number of atoms contributing to the temperature is assumed to be constant for the duration of the run; use the *dynamic* option of the compute_modify command if this is not the case.

The removal of the box deformation velocity component by this fix is essentially computing the temperature after

a "bias" has been removed from the velocity of the atoms. If this compute is used with a fix command that performs thermostatting then this bias will be subtracted from each atom, thermostatting of the remaining thermal velocity will be performed, and the bias will be added back in. Thermostatting fixes that work in this way include fix nvt, fix temp/rescale, fix temp/berendsen, and fix langevin.

This compute subtracts out degrees−of−freedom due to fixes that constrain molecular motion, such as fix shake and fix rigid. This means the temperature of groups of atoms that include these constraints will be computed correctly. If needed, the subtracted degrees−of−freedom can be altered using the *extra* option of the compute_modify command.

See this howto section of the manual for a discussion of different ways to compute temperature and perform thermostatting.

Output info:

This compute calculates a global scalar (the temperature) and a global vector of length 6 (KE tensor), which can be accessed by indices 1−6. These values can be used by any command that uses global scalar or vector values from a compute as input. See this section for an overview of LAMMPS output options.

The scalar value calculated by this compute is "intensive". The vector values are "extensive".

The scalar value will be in temperature units. The vector values will be in energy units.

Restrictions: none

Related commands:

compute temp/ramp, compute temp/profile, fix deform, fix nvt/sllod

Default: none

compute temp/deform/eff command

Syntax:

```
compute ID group-ID temp/deform/eff
```

- ID, group−ID are documented in compute command
- temp/deform/eff = style name of this compute command

Examples:

```
compute myTemp all temp/deform/eff
```

Description:

Define a computation that calculates the temperature of a group of nuclei and electrons in the electron force field model, after subtracting out a streaming velocity induced by the simulation box changing size and/or shape, for example in a non−equilibrium MD (NEMD) simulation. The size/shape change is induced by use of the fix deform/eff command. A compute of this style is created by the fix nvt/sllod/eff command to compute the thermal temperature of atoms for thermostatting purposes. A compute of this style can also be used by any command that computes a temperature, e.g. thermo_modify, fix npt/eff, etc.

The calculation performed by this compute is exactly like that described by the compute temp/deform command, except that the formula for the temperature includes the radial electron velocity contributions, as discussed by the compute temp/eff command. Note that only the translational degrees of freedom for each nuclei or electron are affected by the streaming velocity adjustment. The radial velocity component of the electrons is not affected.

Output info:

This compute calculates a global scalar (the temperature) and a global vector of length 6 (KE tensor), which can be accessed by indices 1−6. These values can be used by any command that uses global scalar or vector values from a compute as input. See this section for an overview of LAMMPS output options.

The scalar value calculated by this compute is "intensive". The vector values are "extensive".

The scalar value will be in temperature units. The vector values will be in energy units.

Restrictions:

This compute is part of the "user−eff" package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

Related commands:

compute temp/ramp, fix deform/eff, fix nvt/sllod/eff

Default: none

compute temp/eff command

Syntax:

```
compute ID group-ID temp/eff
```

- ID, group−ID are documented in compute command
- temp/eff = style name of this compute command

Examples:

```
compute 1 all temp/eff
compute myTemp mobile temp/eff
```

Description:

Define a computation that calculates the temperature of a group of nuclei and electrons in the electron force field model. A compute of this style can be used by commands that compute a temperature, e.g. thermo_modify, fix npt/eff, etc.

The temperature is calculated by the formula KE = dim/2 N k T, where KE = total kinetic energy of the group of atoms (sum of 1/2 m v^2 for nuclei and sum of 1/2 (m v^2 + 3/4 m s^2) for electrons, where s includes the radial electron velocity contributions), dim = 2 or 3 = dimensionality of the simulation, N = number of atoms (only total number of nuclei in the eFF (see the pair_eff command) in the group, k = Boltzmann constant, and T = temperature. This expression is summed over all nuclear and electronic degrees of freedom, essentially by setting the kinetic contribution to the heat capacity to 3/2k (where only nuclei contribute). This subtlety is valid for temperatures well below the Fermi temperature, which for densities two to five times the density of liquid H2 ranges from 86,000 to 170,000 K.

IMPORTANT NOTE: For eFF models, in order to override the default temperature reported by LAMMPS in the thermodynamic quantities reported via the thermo command, the user should apply a thermo_modify command, as shown in the following example:

```
compute         effTemp all temp/eff
thermo_style    custom step etotal pe ke temp press
thermo_modify   temp effTemp
```

A 6−component kinetic energy tensor is also calculated by this compute for use in the computation of a pressure tensor. The formula for the components of the tensor is the same as the above formula, except that v^2 is replaced by vx * vy for the xy component, etc. For the eFF, again, the radial electronic velocities are also considered.

The number of atoms contributing to the temperature is assumed to be constant for the duration of the run; use the *dynamic* option of the compute_modify command if this is not the case.

This compute subtracts out degrees−of−freedom due to fixes that constrain molecular motion, such as fix shake and fix rigid. This means the temperature of groups of atoms that include these constraints will be computed correctly. If needed, the subtracted degrees−of−freedom can be altered using the *extra* option of the compute_modify command.

See this howto section of the manual for a discussion of different ways to compute temperature and perform thermostatting.

Output info:

The scalar value calculated by this compute is "intensive", meaning it is independent of the number of atoms in the simulation. The vector values are "extensive", meaning they scale with the number of atoms in the simulation.

Restrictions:

This compute is part of the "user−eff" package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

Related commands:

compute temp/partial, compute temp/region, compute pressure

Default: none

compute temp/partial command

compute temp/partial/cuda command

Syntax:

```
compute ID group-ID temp/partial xflag yflag zflag
```

- ID, group−ID are documented in compute command
- temp/partial = style name of this compute command
- xflag,yflag,zflag = 0/1 for whether to exclude/include this dimension

Examples:

```
compute newT flow temp/partial 1 1 0
```

Description:

Define a computation that calculates the temperature of a group of atoms, after excluding one or more velocity components. A compute of this style can be used by any command that computes a temperature, e.g. thermo_modify, fix temp/rescale, fix npt, etc.

The temperature is calculated by the formula KE = dim/2 N k T, where KE = total kinetic energy of the group of atoms (sum of 1/2 m v^2), dim = dimensionality of the simulation, N = number of atoms in the group, k = Boltzmann constant, and T = temperature. The calculation of KE excludes the x, y, or z dimensions if xflag, yflag, or zflag = 0. The dim parameter is adjusted to give the correct number of degrees of freedom.

A kinetic energy tensor, stored as a 6−element vector, is also calculated by this compute for use in the calculation of a pressure tensor. The formula for the components of the tensor is the same as the above formula, except that v^2 is replaced by vx*vy for the xy component, etc. The 6 components of the vector are ordered xx, yy, zz, xy, xz, yz.

The number of atoms contributing to the temperature is assumed to be constant for the duration of the run; use the *dynamic* option of the compute_modify command if this is not the case.

The removal of velocity components by this fix is essentially computing the temperature after a "bias" has been removed from the velocity of the atoms. If this compute is used with a fix command that performs thermostatting then this bias will be subtracted from each atom, thermostatting of the remaining thermal velocity will be performed, and the bias will be added back in. Thermostatting fixes that work in this way include fix nvt, fix temp/rescale, fix temp/berendsen, and fix langevin.

This compute subtracts out degrees−of−freedom due to fixes that constrain molecular motion, such as fix shake and fix rigid. This means the temperature of groups of atoms that include these constraints will be computed correctly. If needed, the subtracted degrees−of−freedom can be altered using the *extra* option of the compute_modify command.

See this howto section of the manual for a discussion of different ways to compute temperature and perform thermostatting.

Styles with a *cuda* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in this section of the manual. The accelerated styles take the same arguments and should produce the same results, except for round−off and precision issues.

These accelerated styles are part of the "user−cuda" package. They are only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the −suffix command−line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See this section of the manual for more instructions on how to use the accelerated styles effectively.

Output info:

This compute calculates a global scalar (the temperature) and a global vector of length 6 (KE tensor), which can be accessed by indices 1−6. These values can be used by any command that uses global scalar or vector values from a compute as input. See this section for an overview of LAMMPS output options.

The scalar value calculated by this compute is "intensive". The vector values are "extensive".

The scalar value will be in temperature units. The vector values will be in energy units.

Restrictions: none

Related commands:

compute temp, compute temp/region, compute pressure

Default: none

compute temp/profile command

Syntax:

```
compute ID group-ID temp/profile xflag yflag zflag binstyle args
```

- ID, group−ID are documented in compute command
- temp/profile = style name of this compute command
- xflag,yflag,zflag = 0/1 for whether to exclude/include this dimension
- binstyle = *x* or *y* or *z* or *xy* or *yz* or *xz* or *xyz*

```
  x arg = Nx
    y arg = Ny
    z arg = Nz
    xy args = Nx Ny
    yz args = Ny Nz
    xz args = Nx Nz
    xyz args = Nx Ny Nz
      Nx,Ny,Nz = number of velocity bins in x,y,z dimensions
```

Examples:

```
compute myTemp flow temp/profile 1 1 1 x 10
compute myTemp flow temp/profile 0 1 1 xyz 20 20 20
```

Description:

Define a computation that calculates the temperature of a group of atoms, after subtracting out a spatially−averaged velocity field, before computing the kinetic energy. This can be useful for thermostatting a collection of atoms undergoing a complex flow, e.g. via a profile−unbiased thermostat (PUT) as described in (Evans). A compute of this style can be used by any command that computes a temperature, e.g. thermo_modify, fix temp/rescale, fix npt, etc.

The *xflag*, *yflag*, *zflag* settings determine which components of average velocity are subtracted out.

The *binstyle* setting and its *Nx*, *Ny*, *Nz* arguments determine how bins are setup to perform spatial averaging. "Bins" can be 1d slabs, 2d pencils, or 3d bricks depending on which *binstyle* is used. The simulation box is partitioned conceptually into *Nx* by *Ny* by *Nz* bins. Depending on the *binstyle*, you may only specify one or two of these values; the others are effectively set to 1 (no binning in that dimension). For non−orthogonal (triclinic) simulation boxes, the bins are "tilted" slabs or pencils or bricks that are parallel to the tilted faces of the box. See the region prism command for a discussion of the geometry of tilted boxes in LAMMPS.

When a temperature is computed, the velocity for the set of atoms that are both in the compute group and in the same spatial bin is summed to compute an average velocity for the bin. This bias velocity is then subtracted from the velocities of individual atoms in the bin to yield a thermal velocity for each atom. Note that if there is only one atom in the bin, it's thermal velocity will thus be 0.0.

After the spatially−averaged velocity field has been subtracted from each atom, the temperature is calculated by the formula KE = dim/2 N k T, where KE = total kinetic energy of the group of atoms (sum of 1/2 m v^2), dim = 2 or 3 = dimensionality of the simulation, N = number of atoms in the group, k = Boltzmann constant, and T = temperature.

A kinetic energy tensor, stored as a 6−element vector, is also calculated by this compute for use in the computation of a pressure tensor. The formula for the components of the tensor is the same as the above formula, except that v^2 is replaced by vx*vy for the xy component, etc. The 6 components of the vector are ordered xx, yy, zz, xy, xz, yz.

The number of atoms contributing to the temperature is assumed to be constant for the duration of the run; use the *dynamic* option of the compute_modify command if this is not the case.

The removal of the spatially−averaged velocity field by this fix is essentially computing the temperature after a "bias" has been removed from the velocity of the atoms. If this compute is used with a fix command that performs thermostatting then this bias will be subtracted from each atom, thermostatting of the remaining thermal velocity will be performed, and the bias will be added back in. Thermostatting fixes that work in this way include fix nvt, fix temp/rescale, fix temp/berendsen, and fix langevin.

This compute subtracts out degrees−of−freedom due to fixes that constrain molecular motion, such as fix shake and fix rigid. This means the temperature of groups of atoms that include these constraints will be computed correctly. If needed, the subtracted degrees−of−freedom can be altered using the *extra* option of the compute_modify command.

See this howto section of the manual for a discussion of different ways to compute temperature and perform thermostatting. Using this compute in conjunction with a thermostatting fix, as explained there, will effectively implement a profile−unbiased thermostat (PUT), as described in (Evans).

Output info:

This compute calculates a global scalar (the temperature) and a global vector of length 6 (KE tensor), which can be accessed by indices 1−6. These values can be used by any command that uses global scalar or vector values from a compute as input. See this section for an overview of LAMMPS output options.

The scalar value calculated by this compute is "intensive". The vector values are "extensive".

The scalar value will be in temperature units. The vector values will be in energy units.

Restrictions:

You should not use too large a velocity−binning grid, especially in 3d. In the current implementation, the binned velocity averages are summed across all processors, so this will be inefficient if the grid is too large, and the operation is performed every timestep, as it will be for most thermostats.

Related commands:

compute temp, compute temp/ramp, compute temp/deform, compute pressure

Default:

The option default is units = lattice.

(Evans) Evans and Morriss, Phys Rev Lett, 56, 2172−2175 (1986).

compute temp/ramp command

Syntax:

```
compute ID group-ID temp/ramp vdim vlo vhi dim clo chi keyword value ...
```

- ID, group−ID are documented in compute command
- temp/ramp = style name of this compute command
- vdim = *vx* or *vy* or *vz*
- vlo,vhi = subtract velocities between vlo and vhi (velocity units)
- dim = *x* or *y* or *z*
- clo,chi = lower and upper bound of domain to subtract from (distance units)
- zero or more keyword/value pairs may be appended
- keyword = *units*

units value = *lattice* or *box*

Examples:

```
compute 2nd middle temp/ramp vx 0 8 y 2 12 units lattice
```

Description:

Define a computation that calculates the temperature of a group of atoms, after subtracting out an ramped velocity profile before computing the kinetic energy. A compute of this style can be used by any command that computes a temperature, e.g. thermo_modify, fix temp/rescale, fix npt, etc.

The meaning of the arguments for this command which define the velocity ramp are the same as for the velocity ramp command which was presumably used to impose the velocity.

After the ramp velocity has been subtracted from the specified dimension for each atom, the temperature is calculated by the formula KE = dim/2 N k T, where KE = total kinetic energy of the group of atoms (sum of 1/2 m v^2), dim = 2 or 3 = dimensionality of the simulation, N = number of atoms in the group, k = Boltzmann constant, and T = temperature.

The *units* keyword determines the meaning of the distance units used for coordinates (c1,c2) and velocities (vlo,vhi). A *box* value selects standard distance units as defined by the units command, e.g. Angstroms for units = real or metal. A *lattice* value means the distance units are in lattice spacings; e.g. velocity = lattice spacings / tau. The lattice command must have been previously used to define the lattice spacing.

A kinetic energy tensor, stored as a 6−element vector, is also calculated by this compute for use in the computation of a pressure tensor. The formula for the components of the tensor is the same as the above formula, except that v^2 is replaced by vx*vy for the xy component, etc. The 6 components of the vector are ordered xx, yy, zz, xy, xz, yz.

The number of atoms contributing to the temperature is assumed to be constant for the duration of the run; use the *dynamic* option of the compute_modify command if this is not the case.

The removal of the ramped velocity component by this fix is essentially computing the temperature after a "bias" has been removed from the velocity of the atoms. If this compute is used with a fix command that performs

thermostatting then this bias will be subtracted from each atom, thermostatting of the remaining thermal velocity will be performed, and the bias will be added back in. Thermostatting fixes that work in this way include fix nvt, fix temp/rescale, fix temp/berendsen, and fix langevin.

This compute subtracts out degrees−of−freedom due to fixes that constrain molecular motion, such as fix shake and fix rigid. This means the temperature of groups of atoms that include these constraints will be computed correctly. If needed, the subtracted degrees−of−freedom can be altered using the *extra* option of the compute_modify command.

See this howto section of the manual for a discussion of different ways to compute temperature and perform thermostatting.

Output info:

This compute calculates a global scalar (the temperature) and a global vector of length 6 (KE tensor), which can be accessed by indices 1−6. These values can be used by any command that uses global scalar or vector values from a compute as input. See this section for an overview of LAMMPS output options.

The scalar value calculated by this compute is "intensive". The vector values are "extensive".

The scalar value will be in temperature units. The vector values will be in energy units.

Restrictions: none

Related commands:

compute temp, compute temp/profie, compute temp/deform, compute pressure

Default:

The option default is units = lattice.

compute temp/region command

Syntax:

```
compute ID group-ID temp/region region-ID
```

- ID, group−ID are documented in compute command
- temp/region = style name of this compute command
- region−ID = ID of region to use for choosing atoms

Examples:

```
compute mine flow temp/region boundary
```

Description:

Define a computation that calculates the temperature of a group of atoms in a geometric region. This can be useful for thermostatting one portion of the simulation box. E.g. a McDLT simulation where one side is cooled, and the other side is heated. A compute of this style can be used by any command that computes a temperature, e.g. thermo_modify, fix temp/rescale, etc.

Note that a *region*−style temperature can be used to thermostat with fix temp/rescale or fix langevin, but should probably not be used with Nose/Hoover style fixes (fix nvt, fix npt, or fix nph), if the degrees−of−freedom included in the computed T varies with time.

The temperature is calculated by the formula KE = dim/2 N k T, where KE = total kinetic energy of the group of atoms (sum of 1/2 m v^2), dim = 2 or 3 = dimensionality of the simulation, N = number of atoms in both the group and region, k = Boltzmann constant, and T = temperature.

A kinetic energy tensor, stored as a 6−element vector, is also calculated by this compute for use in the computation of a pressure tensor. The formula for the components of the tensor is the same as the above formula, except that v^2 is replaced by vx*vy for the xy component, etc. The 6 components of the vector are ordered xx, yy, zz, xy, xz, yz.

The number of atoms contributing to the temperature is compute each time the temperature is evaluated since it is assumed atoms can enter/leave the region. Thus there is no need to use the *dynamic* option of the compute_modify command for this compute style.

The removal of atoms outside the region by this fix is essentially computing the temperature after a "bias" has been removed, which in this case is the velocity of any atoms outside the region. If this compute is used with a fix command that performs thermostatting then this bias will be subtracted from each atom, thermostatting of the remaining thermal velocity will be performed, and the bias will be added back in. Thermostatting fixes that work in this way include fix nvt, fix temp/rescale, fix temp/berendsen, and fix langevin. This means any of the thermostatting fixes can operate on a geometric region of atoms, as defined by this compute.

Unlike other compute styles that calculate temperature, this compute does NOT currently subtract out degrees−of−freedom due to fixes that constrain molecular motion, such as fix shake and fix rigid. If needed the subtracted degrees−of−freedom can be altered using the *extra* option of the compute_modify command.

See this howto section of the manual for a discussion of different ways to compute temperature and perform

thermostatting.

Output info:

This compute calculates a global scalar (the temperature) and a global vector of length 6 (KE tensor), which can be accessed by indices 1−6. These values can be used by any command that uses global scalar or vector values from a compute as input. See this section for an overview of LAMMPS output options.

The scalar value calculated by this compute is "intensive". The vector values are "extensive".

The scalar value will be in temperature units. The vector values will be in energy units.

Restrictions: none

Related commands:

compute temp, compute pressure

Default: none

compute temp/region/eff command

Syntax:

```
compute ID group-ID temp/region/eff region-ID
```

- ID, group−ID are documented in compute command
- temp/region/eff = style name of this compute command
- region−ID = ID of region to use for choosing atoms

Examples:

```
compute mine flow temp/region/eff boundary
```

Description:

Define a computation that calculates the temperature of a group of nuclei and electrons in the electron force field model, within a geometric region using the electron force field. A compute of this style can be used by commands that compute a temperature, e.g. thermo_modify.

The operation of this compute is exactly like that described by the compute temp/region command, except that the formula for the temperature itself includes the radial electron velocity contributions, as discussed by the compute temp/eff command.

Output info:

This compute calculates a global scalar (the temperature) and a global vector of length 6 (KE tensor), which can be accessed by indices 1−6. These values can be used by any command that uses global scalar or vector values from a compute as input. See this section for an overview of LAMMPS output options.

The scalar value calculated by this compute is "intensive". The vector values are "extensive".

The scalar value will be in temperature units. The vector values will be in energy units.

Restrictions:

This compute is part of the "user−eff" package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

Related commands:

compute temp/region, compute temp/eff, compute pressure

Default: none

compute temp/rotate command

Syntax:

```
compute ID group-ID temp/rotate
```

- ID, group−ID are documented in compute command
- temp/rotate = style name of this compute command

Examples:

```
compute Tbead bead temp/rotate
```

Description:

Define a computation that calculates the temperature of a group of atoms, after subtracting out the center−of−mass velocity and angular velocity of the group. This is useful if the group is expected to have a non−zero net velocity and/or global rotation motion for some reason. A compute of this style can be used by any command that computes a temperature, e.g. thermo_modify, fix temp/rescale, fix npt, etc.

After the center−of−mass velocity and angular velocity has been subtracted from each atom, the temperature is calculated by the formula KE = dim/2 N k T, where KE = total kinetic energy of the group of atoms (sum of 1/2 m v^2), dim = 2 or 3 = dimensionality of the simulation, N = number of atoms in the group, k = Boltzmann constant, and T = temperature.

A kinetic energy tensor, stored as a 6−element vector, is also calculated by this compute for use in the computation of a pressure tensor. The formula for the components of the tensor is the same as the above formula, except that v^2 is replaced by vx*vy for the xy component, etc. The 6 components of the vector are ordered xx, yy, zz, xy, xz, yz.

The number of atoms contributing to the temperature is assumed to be constant for the duration of the run; use the *dynamic* option of the compute_modify command if this is not the case.

The removal of the center−of−mass velocity and angular velocity by this fix is essentially computing the temperature after a "bias" has been removed from the velocity of the atoms. If this compute is used with a fix command that performs thermostatting then this bias will be subtracted from each atom, thermostatting of the remaining thermal velocity will be performed, and the bias will be added back in. Thermostatting fixes that work in this way include fix nvt, fix temp/rescale, fix temp/berendsen, and fix langevin.

This compute subtracts out degrees−of−freedom due to fixes that constrain molecular motion, such as fix shake and fix rigid. This means the temperature of groups of atoms that include these constraints will be computed correctly. If needed, the subtracted degrees−of−freedom can be altered using the *extra* option of the compute_modify command.

See this howto section of the manual for a discussion of different ways to compute temperature and perform thermostatting.

Output info:

This compute calculates a global scalar (the temperature) and a global vector of length 6 (KE tensor), which can

be accessed by indices 1−6. These values can be used by any command that uses global scalar or vector values from a compute as input. See this section for an overview of LAMMPS output options.

The scalar value calculated by this compute is "intensive". The vector values are "extensive".

The scalar value will be in temperature units. The vector values will be in energy units.

Restrictions:

This compute is part of the "user−misc" package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

Related commands:

compute temp

Default: none

compute temp/sphere command

Syntax:

```
compute ID group-ID temp/sphere keyword value ...
```

- ID, group−ID are documented in compute command
- temp/sphere = style name of this compute command
- zero or more keyword/value pairs may be appended
- keyword = *bias* or *dof*

```
  bias value = bias-IDuniform or gaussian
      bias-ID = ID of a temperature compute that removes a velocity bias
    dof value = all or rotate
      all = compute temperature of translational and rotational degrees of freedom
      rotate = compute temperature of just rotational degrees of freedom
```

Examples:

```
compute 1 all temp/sphere
compute myTemp mobile temp/sphere bias tempCOM
compute myTemp mobile temp/sphere dof rotate
```

Description:

Define a computation that calculates the temperature of a group of spherical particles, including a contribution from both their translational and rotational kinetic energy. This differs from the usual compute temp command, which assumes point particles with only translational kinetic energy.

Both point and finite−size particles can be included in the group. Point particles do not rotate, so they have only 3 translational degrees of freedom. For 3d spherical particles, each has 6 degrees of freedom (3 translational, 3 rotational). For 2d spherical particles, each has 3 degrees of freedom (2 translational, 1 rotational).

IMPORTANT NOTE: This choice for degrees of freedom (dof) assumes that all finite−size spherical particles in your model will freely rotate, sampling all their rotational dof. It is possible to use a combination of interaction potentials and fixes that induce no torque or otherwise constrain some of all of your particles so that this is not the case. Then there are less dof and you should use the compute_modify extra command to adjust the dof accordingly.

The translational kinetic energy is computed the same as is described by the compute temp command. The rotational kinetic energy is computed as 1/2 I w^2, where I is the moment of inertia for a sphere and w is the particle's angular velocity.

IMPORTANT NOTE: For 2d models, particles are treated as spheres, not disks, meaning their moment of inertia will be the same as in 3d.

A kinetic energy tensor, stored as a 6−element vector, is also calculated by this compute. The formula for the components of the tensor is the same as the above formulas, except that v^2 and w^2 are replaced by vx*vy and wx*wy for the xy component. The 6 components of the vector are ordered xx, yy, zz, xy, xz, yz.

The number of atoms contributing to the temperature is assumed to be constant for the duration of the run; use the *dynamic* option of the compute_modify command if this is not the case.

This compute subtracts out translational degrees−of−freedom due to fixes that constrain molecular motion, such as fix shake and fix rigid. This means the temperature of groups of atoms that include these constraints will be computed correctly. If needed, the subtracted degrees−of−freedom can be altered using the *extra* option of the compute_modify command.

See this howto section of the manual for a discussion of different ways to compute temperature and perform thermostatting.

The keyword/value option pairs are used in the following ways.

For the *bias* keyword, *bias−ID* refers to the ID of a temperature compute that removes a "bias" velocity from each atom. This allows compute temp/sphere to compute its thermal temperature after the translational kinetic energy components have been altered in a prescribed way, e.g. to remove a velocity profile. Thermostats that use this compute will work with this bias term. See the doc pages for individual computes that calculate a temperature and the doc pages for fixes that perform thermostatting for more details.

For the *dof* keyword, a setting of *all* calculates a temperature that includes both translational and rotational degrees of freedom. A setting of *rotate* calculates a temperature that includes only rotational degrees of freedom.

Output info:

This compute calculates a global scalar (the temperature) and a global vector of length 6 (KE tensor), which can be accessed by indices 1−6. These values can be used by any command that uses global scalar or vector values from a compute as input. See this section for an overview of LAMMPS output options.

The scalar value calculated by this compute is "intensive". The vector values are "extensive".

The scalar value will be in temperature units. The vector values will be in energy units.

Restrictions:

This fix requires that atoms store torque and angular velocity (omega) and a radius as defined by the atom_style sphere command.

All particles in the group must be finite−size spheres, or point particles with radius = 0.0.

Related commands:

compute temp, compute temp/asphere

Default:

The option defaults are no bias and dof = all.

compute ti command

Syntax:

```
compute ID group ti keyword args ...
```

- ID, group−ID are documented in compute command
- ti = style name of this compute command
- one or more attribute/arg pairs may be appended
- keyword = pair style (lj/cut, gauss, born, etc) or *tail* or *kspace*

```
pair style args = v_name1 v_name2
  v_name1 = variable with name1 that is energy scale factor and function of lambda
  v_name2 = variable with name2 that is derivative of v_name1 with respect to lambda
tail args = v_name1 v_name2
  v_name1 = variable with name1 that is energy tail correction scale factor and function of
  v_name2 = variable with name2 that is derivative of v_name1 with respect to lambda
kspace args = v_name1 v_name2
  v_name1 = variable with name1 that is K−Space scale factor and function of lambda
  v_name2 = variable with name2 that is derivative of v_name1 with respect to lambda
```

Examples:

```
compute 1 all ti lj/cut v_lj v_dlj coul/long v_c v_dc kspace v_ks v_dks
```

Description:

Define a computation that calculates the derivative of the interaction potential with respect to *lambda*, the coupling parameter used in a thermodynamic integration. This derivative can be used to infer a free energy difference resulting from an alchemical simulation, as described in Eike.

Typically this compute will be used in conjunction with the fix adapt command which can perform alchemical transformations by adusting the strength of an interaction potential as a simulation runs, as defined by one or more pair_style or kspace_style commands. This scaling is done via a prefactor on the energy, forces, virial calculated by the pair or K−Space style. The prefactor is often a function of a *lambda* parameter which may be adjusted from 0 to 1 (or vice versa) over the course of a run. The time−dependent adjustment is what the fix adapt command does.

Assume that the unscaled energy of a pair_style or kspace_style is given by U. Then the scaled energy is

```
Us = f(lambda) U
```

where f() is some function of lambda. What this compute calculates is

```
dUs / d(lambda) = U df(lambda)/dlambda = Us / f(lambda) df(lambda)/dlambda
```

which is the derivative of the system's scaled potential energy Us with respect to *lambda*.

To do this calculation, you provide two functions, as equal−style variables. The first is specified as *v_name1*, where *name1* is the name of the variable, and is f(lambda) in the notation above. The second is specified as *v_name2*, where *name2* is the name of the variable, and is df(lambda) / dlambda in the notation above. I.e. it is the analytic derivative of f() with respect to lambda. Note that the *name1* variable is also typically given as an

argument to the fix adapt command.

An alchemical simulation may use several pair potentials together, invoked via the pair_style hybrid or hybrid/overlay command. The total dUs/dlambda for the overall system is calculated as the sum of each contributing term as listed by the keywords in the compute ti command. Individual pair potentials can be listed, which will be sub−styles in the hybrid case. You can also include a K−space term via the *kspace* keyword. You can also include a pairwise long−range tail correction to the energy via the *tail* keyword.

For each term you can specify a different (or the same) scale factor by the two variables that you list. Again, these will typically correspond toe the scale factors applied to these various potentials and the K−Space contribution via the fix_adapt command.

More details about the exact functional forms for the computation of du/dl can be found in the paper by Eike.

Output info:

This compute calculates a global scalar, namely dUs/dlambda. This value can be used by any command that uses a global scalar value from a compute as input. See this section for an overview of LAMMPS output options.

The scalar value calculated by this compute is "extensive".

The scalar value will be in energy units.

Restrictions: none

Related commands:

fix adapt

Default: none

(Eike) Eike and Maginn, Journal of Chemical Physics, 124, 164503 (2006).

create_atoms command

Syntax:

```
create_atoms type style args keyword values ...
```

- type = atom type (1−Ntypes) of atoms to create
- style = *box* or *region* or *single* or *random*

```
  box args = none
    region args = region-ID
      region-ID = atoms will only be created if contained in the region
    single args = x y z
      x,y,z = coordinates of a single atom (distance units)
    random args = N seed region-ID
      N = number of atoms to create
      seed = random # seed (positive integer)
      region-ID = create atoms within this region, use NULL for entire simulation box
```

- zero or more keyword/value pairs may be appended
- keyword = *basis* or *units*

```
  basis values = M itype
      M = which basis atom
      itype = atom type (1-N) to assign to this basis atom
    units value = lattice or box
      lattice = the geometry is defined in lattice units
      box = the geometry is defined in simulation box units
```

Examples:

```
create_atoms 1 box
create_atoms 3 region regsphere basis 2 3
create_atoms 3 single 0 0 5
```

Description:

This command creates atoms on a lattice, or a single atom, or a random collection of atoms, as an alternative to reading in their coordinates explicitly via a read_data or read_restart command. A simulation box must already exist, which is typically created via the create_box command. Before using this command, a lattice must also be defined using the lattice command. The only exceptions are for the *single* style with units = box or the *random* style.

For the *box* style, the create_atoms command fills the entire simulation box with atoms on the lattice. If your simulation box is periodic, you should insure its size is a multiple of the lattice spacings, to avoid unwanted atom overlaps at the box boundaries. If your box is periodic and a multiple of the lattice spacing in a particular dimension, LAMMPS is careful to put exactly one atom at the boundary (on either side of the box), not zero or two.

For the *region* style, the geometric volume is filled that is inside the simulation box and is also consistent with the region volume. See the region command for details. Note that a region can be specified so that its "volume" is either inside or outside a geometric boundary. Also note that if your region is the same size as a periodic simulation box (in some dimension), LAMMPS does not implement the same logic as with the *box* style, to insure exactly one atom at the boundary. if this is what you desire, you should either use the *box* style, or tweak the

region size to get precisely the atoms you want.

For the *single* style, a single atom is added to the system at the specified coordinates. This can be useful for debugging purposes or to create a tiny system with a handful of atoms at specified positions.

For the *random* style, N atoms are added to the system at randomly generated coordinates, which can be useful for generating an amorphous system. The atoms are created one by one using the speficied random number *seed*, resulting in the same set of atom coordinates, independent of how many processors are being used in the simulation. If the *region−ID* argument is specified as NULL, then the created atoms will be anywhere in the simulation box. If a *region−ID* is specified, a geometric volume is filled that is inside the simulation box and is also consistent with the region volume. See the region command for details. Note that a region can be specified so that its "volume" is either inside or outside a geometric boundary.

IMPORTANT NOTE: The atoms generated by the *random* style will typically be highly overlapped which will cause many interatomic potentials to compute large energies and forces. Thus you should either perform an energy minimization or run dynamics with fix nve/limit to equilibrate such a system, before running normal dynamics.

The *basis* keyword specifies an atom type that will be assigned to specific basis atoms as they are created. See the lattice command for specifics on how basis atoms are defined for the unit cell of the lattice. By default, all created atoms are assigned the argument *type* as their atom type.

The *units* keyword determines the meaning of the distance units used to specify the coordinates of the one atom created by the *single* style. A *box* value selects standard distance units as defined by the units command, e.g. Angstroms for units = real or metal. A *lattice* value means the distance units are in lattice spacings.

Note that this command adds atoms to those that already exist. By using the create_atoms command multiple times, multiple sets of atoms can be added to the simulation. For example, interleaving create_atoms with lattice commands specifying different orientations, grain boundaries can be created. By using the create_atoms command in conjunction with the delete_atoms command, reasonably complex geometries can be created. The create_atoms command can also be used to add atoms to a system previously read in from a data or restart file. In all these cases, care should be taken to insure that new atoms do not overlap existing atoms inappropriately. The delete_atoms command can be used to handle overlaps.

Atom IDs are assigned to created atoms in the following way. The collection of created atoms are assigned consecutive IDs that start immediately following the largest atom ID existing before the create_atoms command was invoked. When a simulation is performed on different numbers of processors, there is no guarantee a particular created atom will be assigned the same ID.

Aside from their ID, atom type, and xyz position, other properties of created atoms are set to default values, depending on which quantities are defined by the chosen atom style. See the atom style command for more details. See the set and velocity commands for info on how to change these values.

- charge = 0.0
- dipole moment magnitude = 0.0
- diameter = 1.0
- shape = 0.0 0.0 0.0
- density = 1.0
- volume = 1.0
- velocity = 0.0 0.0 0.0
- angular velocity = 0.0 0.0 0.0
- angular momentum = 0.0 0.0 0.0

- quaternion = (1,0,0,0)
- bonds, angles, dihedrals, impropers = none

Note that the *sphere* atom style sets the default particle diameter to 1.0 as well as the density. This means the mass for the particle is not 1.0, but is PI/6 * diameter^3 = 0.5236.

Note that the *ellipsoid* atom style sets the default particle shape to (0.0 0.0 0.0) and the density to 1.0 which means it is a point particle, not an ellipsoid, and has a mass of 1.0.

Note that the *peri* style sets the default volume and density to 1.0 and thus also set the mass for the particle to 1.0.

The set command can be used to override many of these default settings.

Restrictions:

An atom_style must be previously defined to use this command.

Related commands:

lattice, region, create_box, read_data, read_restart

Default:

The default for the *basis* keyword, is all created atoms are assigned the argument *type* as their atom type. The default for the *units* keyword is lattice.

create_box command

Syntax:

```
create_box N region-ID
```

- N = # of atom types to use in this simulation
- region−ID = ID of region to use as simulation domain

Examples:

```
create_box 2 mybox
```

Description:

This command creates a simulation box based on the specified region. Thus a region command must first be used to define a geometric domain.

The argument N is the number of atom types that will be used in the simulation.

If the region is not of style *prism*, then LAMMPS encloses the region (block, sphere, etc) with an axis−aligned orthogonal bounding box which becomes the simulation domain.

If the region is of style *prism*, LAMMPS creates a non−orthogonal simulation domain shaped as a parallelepiped with triclinic symmetry. As defined by the region prism command, the parallelepiped has its "origin" at (xlo,ylo,zlo) and is defined by 3 edge vectors starting from the origin given by A = (xhi−xlo,0,0); B = (xy,yhi−ylo,0); C = (xz,yz,zhi−zlo). *Xy*,*xz*,*yz* can be 0.0 or positive or negative values and are called "tilt factors" because they are the amount of displacement applied to faces of an originally orthogonal box to transform it into the parallelipiped.

A *prism* region used with the create_box command must have tilt factors (xy,xz,yz) that do not skew the box more than half the distance of the parallel box length. For example, if xlo = 2 and xhi = 12, then the x box length is 10 and the xy tilt factor must be between −5 and 5. Similarly, both xz and yz must be between −(xhi−xlo)/2 and +(yhi−ylo)/2. Note that this is not a limitation, since if the maximum tilt factor is 5 (as in this example), then configurations with tilt = ..., −15, −5, 5, 15, 25, ... are all geometrically equivalent.

See this section of the doc pages for a geometric description of triclinic boxes, as defined by LAMMPS, and how to transform these parameters to and from other commonly used triclinic representations.

When a prism region is used, the simulation domain must be periodic in any dimensions with a non−zero tilt factor, as defined by the boundary command. I.e. if the xy tilt factor is non−zero, then both the x and y dimensions must be periodic. Similarly, x and z must be periodic if xz is non−zero and y and z must be periodic if yz is non−zero. Also note that if your simulation will tilt the box, e.g. via the fix deform command, the simulation box must be defined as triclinic, even if the tilt factors are initially 0.0.

IMPORTANT NOTE: If the system is non−periodic (in a dimension), then you should not make the lo/hi box dimensions (as defined in your region command) radically smaller/larger than the extent of the atoms you eventually plan to create, e.g. via the create_atoms command. For example, if your atoms extend from 0 to 50, you should not specify the box bounds as −10000 and 10000. This is because LAMMPS uses the specified box size to layout the 3d grid of processors. A huge (mostly empty) box will be sub−optimal for performance when

using "fixed" boundary conditions (see the boundary command). When using "shrink−wrap" boundary conditions (see the boundary command), a huge (mostly empty) box may cause a parallel simulation to lose atoms the first time that LAMMPS shrink−wraps the box around the atoms.

Restrictions:

An atom_style and region must have been previously defined to use this command.

Related commands:

create_atoms, region

Default: none

delete_atoms command

Syntax:

```
delete_atoms style args keyword value ...
```

- style = *group* or *region* or *overlap* or *porosity*

```
  group args = group-ID
    region args = region-ID
    overlap args = cutoff group1-ID group2-ID
      cutoff = delete one atom from pairs of atoms within the cutoff (distance units)
      group1-ID = one atom in pair must be in this group
      group2-ID = other atom in pair must be in this group
    porosity args = region-ID fraction seed
      region-ID = region within which to perform deletions
      fraction = delete this fraction of atoms
      seed = random number seed (positive integer)
```

- zero or more keyword/value pairs may be appended
- keyword = *compress*

```
  compress value = no or yes
```

Examples:

```
delete_atoms group edge
delete_atoms region sphere compress no
delete_atoms overlap 0.3 all all
delete_atoms overlap 0.5 solvent colloid
delete_atoms porosity cube 0.1 482793
```

Description:

Delete the specified atoms. This command can be used to carve out voids from a block of material or to delete created atoms that are too close to each other (e.g. at a grain boundary).

For style *group*, all atoms belonging to the group are deleted.

For style *region*, all atoms in the region volume are deleted.

For style *overlap* pairs of atoms whose distance of separation is within the specified cutoff distance are searched for, and one of the 2 atoms is deleted. Only pairs where one of the two atoms is in the first group specified and the other atom is in the second group are considered. The atom that is in the first group is the one that is deleted.

Note that it is OK for the two group IDs to be the same (e.g. group *all*), or for some atoms to be members of both groups. In these cases, either atom in the pair may be deleted. Also note that if there are atoms which are members of both groups, the only guarantee is that at the end of the deletion operation, enough deletions will have occurred that no atom pairs within the cutoff will remain (subject to the group restriction). There is no guarantee that the minimum number of atoms will be deleted, or that the same atoms will be deleted when running on different numbers of processors.

For style *porosity* a specified *fraction* of atoms are deleted within the specified region. For example, if fraction is 0.1, then 10% of the atoms will be deleted. The atoms to delete are chosen randomly. There is no guarantee that

the exact fraction of atoms will be deleted, or that the same atoms will be deleted when running on different numbers of processors.

If the *compress* keyword is set to *yes*, then after atoms are deleted, then atom IDs are re−assigned so that they run from 1 to the number of atoms in the system. This is not done for molecular systems (see the atom_style command), regardless of the *compress* setting, since it would foul up the bond connectivity that has already been assigned.

Restrictions:

The *overlap* styles requires inter−processor communication to acquire ghost atoms and build a neighbor list. This means that your system must be ready to perform a simulation before using this command (force fields setup, atom masses set, etc). Since a neighbor list is used to find overlapping atom pairs, it also means that you must define a pair style with force cutoffs greater than or equal to the desired overlap cutoff between pairs of relevant atom types, even though the pair potential will not be evaluated.

If the special_bonds command is used with a setting of 0, then a pair of bonded atoms (1−2, 1−3, or 1−4) will not appear in the neighbor list, and thus will not be considered for deletion by the *overlap* styles. You probably don't want to be deleting one atom in a bonded pair anyway.

Related commands:

create_atoms

Default:

The option defaults are compress = yes.

delete_bonds command

Syntax:

```
delete_bonds group-ID style args keyword ...
```

- group−ID = group ID
- style = *multi* or *atom* or *bond* or *angle* or *dihedral* or *improper* or *stats*

```
  multi args = none
    atom args = an atom type
    bond args = a bond type
    angle args = an angle type
    dihedral args = a dihedral type
    improper args = an improper type
    stats args = none
```

- zero or more keywords may be appended
- keyword = *undo* or *remove* or *special*

Examples:

```
delete_bonds frozen multi remove
delete_bonds all atom 4 special
delete_bonds all stats
```

Description:

Turn off (or on) molecular topology interactions, i.e. bonds, angles, dihedrals, impropers. This command is useful for deleting interactions that have been previously turned off by bond−breaking potentials. It is also useful for turning off topology interactions between frozen or rigid atoms. Pairwise interactions can be turned off via the neigh_modify exclude command. The fix shake command also effectively turns off certain bond and angle interactions.

For all styles, an interaction is only turned off (or on) if all the atoms involved are in the specified group. For style *multi* this is the only criterion applied − all types of bonds, angles, dihedrals, impropers in the group turned off.

For style *atom*, one or more of the atoms involved must also be of the specified type. For style *bond*, only bonds are candidates for turn−off, and the bond must be of the specified type. Styles *angle*, *dihedral*, and *improper* are treated similarly.

For style *bond*, you can set the type to 0 to delete bonds that have been previously broken; e.g. see the bond_style quartic command.

For style *stats* no interactions are turned off (or on); the status of all interactions in the specified group is simply reported. This is useful for diagnostic purposes if bonds have been turned off by a bond−breaking potential during a previous run.

The default behavior of the delete_bonds command is to turn off interactions by toggling their type to a negative value. E.g. a bond_type of 2 is set to −2. The neighbor list creation routines will not include such an interaction in their interaction lists. The default is also to not alter the list of 1−2, 1−3, 1−4 neighbors computed by the special_bonds command and used to weight pairwise force and energy calculations. This means that pairwise computations will proceed as if the bond (or angle, etc) were still turned on.

The keywords listed above can be appended to the argument list to alter the default behavior.

The *undo* keyword inverts the delete_bonds command so that the specified bonds, angles, etc are turned on if they are currently turned off. This means any negative value is toggled to positive. Note that the fix shake command also sets bond and angle types negative, so this option should not be used on those interactions.

The *remove* keyword is invoked at the end of the delete_bonds operation. It causes turned−off bonds (angles, etc) to be removed from each atom's data structure and then adjusts the global bond (angle, etc) counts accordingly. Removal is a permanent change; removed bonds cannot be turned back on via the *undo* keyword. Removal does not alter the pairwise 1−2, 1−3, 1−4 weighting list.

The *special* keyword is invoked at the end of the delete_bonds operation, after (optional) removal. It re−computes the pairwise 1−2, 1−3, 1−4 weighting list. The weighting list computation treats turned−off bonds the same as turned−on. Thus, turned−off bonds must be removed if you wish to change the weighting list.

Note that the choice of *remove* and *special* options affects how 1−2, 1−3, 1−4 pairwise interactions will be computed across bonds that have been modified by the delete_bonds command.

Restrictions:

This command requires inter−processor communication to coordinate the deleting of bonds. This means that your system must be ready to perform a simulation before using this command (force fields setup, atom masses set, etc).

If deleted bonds (angles, etc) are removed but the 1−2, 1−3, 1−4 weighting list is not recomputed, this can cause a later fix shake command to fail due to an atom's bonds being inconsistent with the weighting list. This should only happen if the group used in the fix command includes both atoms in the bond, in which case you probably should be recomputing the weighting list.

Related commands:

neigh_modify exclude, special_bonds, fix shake

Default: none

dielectric command

Syntax:

```
dielectric value
```

- value = dielectric constant

Examples:

```
dielectric 2.0
```

Description:

Set the dielectric constant for Coulombic interactions (pairwise and long−range) to this value. The constant is unitless, since it is used to reduce the strength of the interactions. The value is used in the denominator of the formulas for Coulombic interactions − e.g. a value of 4.0 reduces the Coulombic interactions to 25% of their default strength. See the pair_style command for more details.

Restrictions: none

Related commands:

pair_style

Default:

```
dielectric 1.0
```

dihedral_style charmm command

Syntax:

```
dihedral_style charmm
```

Examples:

```
dihedral_style charmm
dihedral_coeff 1 120.0 1 60 0.5
```

Description:

The *charmm* dihedral style uses the potential

$$E = K[1 + \cos(n\phi - d)]$$

See (MacKerell) for a description of the CHARMM force field. This dihedral style can also be used for the AMBER force field (see comment on weighting factors below). See (Cornell) for a description of the AMBER force field.

The following coefficients must be defined for each dihedral type via the dihedral_coeff command as in the example above, or in the data file or restart files read by the read_data or read_restart commands:

- K (energy)
- n (integer >= 0)
- d (integer value of degrees)
- weighting factor (0.0 to 1.0)

The weighting factor is applied to pairwise interaction between the 1st and 4th atoms in the dihedral, which are computed by a CHARMM pair_style with epsilon and sigma values specified with a pair_coeff command. Note that this weighting factor is unrelated to the weighting factor specified by the special bonds command which applies to all 1−4 interactions in the system.

For CHARMM force fields, the special_bonds 1−4 weighting factor should be set to 0.0. This is because the pair styles that contain "charmm" (e.g. pair_style lj/charmm/coul/long) define extra 1−4 interaction coefficients that are used by this dihedral style to compute those interactions explicitly. This means that if any of the weighting factors defined as dihedral coefficients (4th coeff above) are non−zero, then you must use a charmm pair style. Note that if you do not set the special_bonds 1−4 weighting factor to 0.0 (which is the default) then 1−4 interactions in dihedrals will be computed twice, once by the pair routine and once by the dihedral routine, which is probably not what you want.

For AMBER force fields, the special_bonds 1−4 weighting factor should be set to the AMBER defaults (1/2 and 5/6) and all the dihedral weighting factors (4th coeff above) should be set to 0.0. In this case, you can use any pair style you wish, since the dihedral does not need any 1−4 information.

Restrictions:

This dihedral style can only be used if LAMMPS was built with the "molecular" package (which it is by default). See the Making LAMMPS section for more info on packages.

Related commands:

dihedral_coeff

Default: none

(Cornell) Cornell, Cieplak, Bayly, Gould, Merz, Ferguson, Spellmeyer, Fox, Caldwell, Kollman, JACS 117, 5179−5197 (1995).

(MacKerell) MacKerell, Bashford, Bellott, Dunbrack, Evanseck, Field, Fischer, Gao, Guo, Ha, et al, J Phys Chem B, 102, 3586 (1998).

dihedral_style class2 command

Syntax:

```
dihedral_style class2
```

Examples:

```
dihedral_style class2
dihedral_coeff 1 100 75 100 70 80 60
dihedral_coeff * mbt 3.5945 0.1704 −0.5490 1.5228
dihedral_coeff * ebt 0.3417 0.3264 −0.9036 0.1368 0.0 −0.8080 1.0119 1.1010
dihedral_coeff 2 at 0.0 −0.1850 −0.7963 −2.0220 0.0 −0.3991 110.2453 105.1270
dihedral_coeff * aat −13.5271 110.2453 105.1270
dihedral_coeff * bb13 0.0 1.0119 1.1010
```

Description:

The *class2* dihedral style uses the potential

$$
\begin{aligned}
E &= E_d + E_{mbt} + E_{ebt} + E_{at} + E_{aat} + E_{bb13} \\
E_d &= \sum_{n=1}^{3} K_n[1 - \cos(n\phi - \phi_n)] \\
E_{mbt} &= (r_{jk} - r_2)[A_1 \cos(\phi) + A_2 \cos(2\phi) + A_3 \cos(3\phi)] \\
E_{ebt} &= (r_{ij} - r_1)[B_1 \cos(\phi) + B_2 \cos(2\phi) + B_3 \cos(3\phi)] + \\
&\quad (r_{kl} - r_3)[C_1 \cos(\phi) + C_2 \cos(2\phi) + C_3 \cos(3\phi)] \\
E_{at} &= (\theta_{ijk} - \theta_1)[D_1 \cos(\phi) + D_2 \cos(2\phi) + D_3 \cos(3\phi)] + \\
&\quad (\theta_{jkl} - \theta_2)[E_1 \cos(\phi) + E_2 \cos(2\phi) + E_3 \cos(3\phi)] \\
E_{aat} &= M(\theta_{ijk} - \theta_1)(\theta_{jkl} - \theta_2) \cos(\phi) \\
E_{bb13} &= N(r_{ij} - r_1)(r_{kl} - r_3)
\end{aligned}
$$

where Ed is the dihedral term, Embt is a middle−bond−torsion term, Eebt is an end−bond−torsion term, Eat is an angle−torsion term, Eaat is an angle−angle−torsion term, and Ebb13 is a bond−bond−13 term.

Theta1 and theta2 are equilibrium angles and r1 r2 r3 are equilibrium bond lengths.

See (Sun) for a description of the COMPASS class2 force field.

Coefficients for the Ed, Embt, Eebt, Eat, Eaat, and Ebb13 formulas must be defined for each dihedral type via the dihedral_coeff command as in the example above, or in the data file or restart files read by the read_data or read_restart commands.

These are the 6 coefficients for the Ed formula:

- K1 (energy)
- phi1 (degrees)
- K2 (energy)

- phi2 (degrees)
- K3 (energy)
- phi3 (degrees)

For the Embt formula, each line in a dihedral_coeff command in the input script lists 5 coefficients, the first of which is "mbt" to indicate they are MiddleBondTorsion coefficients. In a data file, these coefficients should be listed under a "MiddleBondTorsion Coeffs" heading and you must leave out the "mbt", i.e. only list 4 coefficients after the dihedral type.

- mbt
- A1 (energy/distance)
- A2 (energy/distance)
- A3 (energy/distance)
- r2 (distance)

For the Eebt formula, each line in a dihedral_coeff command in the input script lists 9 coefficients, the first of which is "ebt" to indicate they are EndBondTorsion coefficients. In a data file, these coefficients should be listed under a "EndBondTorsion Coeffs" heading and you must leave out the "ebt", i.e. only list 8 coefficients after the dihedral type.

- ebt
- B1 (energy/distance)
- B2 (energy/distance)
- B3 (energy/distance)
- C1 (energy/distance)
- C2 (energy/distance)
- C3 (energy/distance)
- r1 (distance)
- r3 (distance)

For the Eat formula, each line in a dihedral_coeff command in the input script lists 9 coefficients, the first of which is "at" to indicate they are AngleTorsion coefficients. In a data file, these coefficients should be listed under a "AngleTorsion Coeffs" heading and you must leave out the "at", i.e. only list 8 coefficients after the dihedral type.

- at
- D1 (energy/radian)
- D2 (energy/radian)
- D3 (energy/radian)
- E1 (energy/radian)
- E2 (energy/radian)
- E3 (energy/radian)
- theta1 (degrees)
- theta2 (degrees)

Theta1 and theta2 are specified in degrees, but LAMMPS converts them to radians internally; hence the units of D and E are in energy/radian.

For the Eaat formula, each line in a dihedral_coeff command in the input script lists 4 coefficients, the first of which is "aat" to indicate they are AngleAngleTorsion coefficients. In a data file, these coefficients should be listed under a "AngleAngleTorsion Coeffs" heading and you must leave out the "aat", i.e. only list 3 coefficients after the dihedral type.

- aat
- M (energy/radian^2)
- theta1 (degrees)
- theta2 (degrees)

Theta1 and theta2 are specified in degrees, but LAMMPS converts them to radians internally; hence the units of M are in energy/radian^2.

For the Ebb13 formula, each line in a dihedral_coeff command in the input script lists 4 coefficients, the first of which is "bb13" to indicate they are BondBond13 coefficients. In a data file, these coefficients should be listed under a "BondBond13 Coeffs" heading and you must leave out the "bb13", i.e. only list 3 coefficients after the dihedral type.

- bb13
- N (energy/distance^2)
- r1 (distance)
- r3 (distance)

Restrictions:

This dihedral style can only be used if LAMMPS was built with the "class2" package. See the Making LAMMPS section for more info on packages.

Related commands:

dihedral_coeff

Default: none

(Sun) Sun, J Phys Chem B 102, 7338−7364 (1998).

dihedral_coeff command

Syntax:

```
dihedral_coeff N args
```

- N = dihedral type (see asterisk form below)
- args = coefficients for one or more dihedral types

Examples:

```
dihedral_coeff 1 80.0 1 3
dihedral_coeff * 80.0 1 3 0.5
dihedral_coeff 2* 80.0 1 3 0.5
```

Description:

Specify the dihedral force field coefficients for one or more dihedral types. The number and meaning of the coefficients depends on the dihedral style. Dihedral coefficients can also be set in the data file read by the read_data command or in a restart file.

N can be specified in one of two ways. An explicit numeric value can be used, as in the 1st example above. Or a wild−card asterisk can be used to set the coefficients for multiple dihedral types. This takes the form "*" or "*n" or "n*" or "m*n". If N = the number of dihedral types, then an asterisk with no numeric values means all types from 1 to N. A leading asterisk means all types from 1 to n (inclusive). A trailing asterisk means all types from n to N (inclusive). A middle asterisk means all types from m to n (inclusive).

Note that using a dihedral_coeff command can override a previous setting for the same dihedral type. For example, these commands set the coeffs for all dihedral types, then overwrite the coeffs for just dihedral type 2:

```
dihedral_coeff * 80.0 1 3
dihedral_coeff 2 200.0 1 3
```

A line in a data file that specifies dihedral coefficients uses the exact same format as the arguments of the dihedral_coeff command in an input script, except that wild−card asterisks should not be used since coefficients for all N types must be listed in the file. For example, under the "Dihedral Coeffs" section of a data file, the line that corresponds to the 1st example above would be listed as

```
1 80.0 1 3
```

The dihedral_style class2 is an exception to this rule, in that an additional argument is used in the input script to allow specification of the cross−term coefficients. See its doc page for details.

Here is an alphabetic list of dihedral styles defined in LAMMPS. Click on the style to display the formula it computes and coefficients specified by the associated dihedral_coeff command:

- dihedral_style none − turn off dihedral interactions
- dihedral_style hybrid − define multiple styles of dihedral interactions

- dihedral_style charmm − CHARMM dihedral
- dihedral_style class2 − COMPASS (class 2) dihedral

- dihedral_style harmonic − harmonic dihedral
- dihedral_style helix − helix dihedral
- dihedral_style multi/harmonic − multi−harmonic dihedral
- dihedral_style opls − OPLS dihedral

There are also additional dihedral styles submitted by users which are included in the LAMMPS distribution. The list of these with links to the individual styles are given in the dihedral section of this page.

Restrictions:

This command must come after the simulation box is defined by a read_data, read_restart, or create_box command.

A dihedral style must be defined before any dihedral coefficients are set, either in the input script or in a data file.

Related commands:

dihedral_style

Default: none

dihedral_style cosine/shift/exp command

Syntax:

```
dihedral_style cosine/shift/exp
```

Examples:

```
dihedral_style cosine/shift/exp
dihedral_coeff 1 10.0 45.0 2.0
```

Description:

The *cosine/shift/exp* dihedral style uses the potential

$$E = -U_{min}\frac{e^{-aU(\theta,\theta_0)}-1}{e^{a}-1} \quad \text{with} \quad U(\theta,\theta_0) = -0.5\left(1+\cos(\theta-\theta_0)\right)$$

where Umin, theta, and a are defined for each dihedral type.

The potential is bounded between [−Umin:0] and the minimum is located at the angle theta0. The a parameter can be both positive or negative and is used to control the spring constant at the equilibrium.

The spring constant is given by k=a exp(a) Umin/ [2 (Exp(a)−1)]. For a>3 k/Umin = a/2 to better than 5% relative error. For negative values of the a parameter, the spring constant is essentially zero, and anharmonic terms takes over. The potential is furthermore well behaved in the limit a−>0, where it has been implemented to linear order in a for a < 0.001.

The following coefficients must be defined for each dihedral type via the dihedral_coeff command as in the example above, or in the data file or restart files read by the read_data or read_restart commands:

- umin (energy)
- theta (angle)
- A (real number)

Restrictions:

This dihedral style can only be used if LAMMPS was built with the "user−misc" package. See the Making LAMMPS section for more info on packages.

Related commands:

dihedral_coeff, angle_cosineshiftexp

Default: none

dihedral_style harmonic command

Syntax:

```
dihedral_style harmonic
```

Examples:

```
dihedral_style harmonic
dihedral_coeff 1 80.0 1 2
```

Description:

The *harmonic* dihedral style uses the potential

$$E = K[1 + d\cos(n\phi)]$$

The following coefficients must be defined for each dihedral type via the dihedral_coeff command as in the example above, or in the data file or restart files read by the read_data or read_restart commands:

- K (energy)
- d (+1 or −1)
- n (integer >= 0)

Restrictions:

This dihedral style can only be used if LAMMPS was built with the "molecular" package (which it is by default). See the Making LAMMPS section for more info on packages.

Related commands:

dihedral_coeff

Default: none

dihedral_style helix command

Syntax:

```
dihedral_style helix
```

Examples:

```
dihedral_style helix
dihedral_coeff 1 80.0 100.0 40.0
```

Description:

The *helix* dihedral style uses the potential

$$E = A[1 - \cos(\theta)] + B[1 + \cos(3\theta)] + C[1 + \cos(\theta + \frac{\pi}{4})]$$

This coarse−grain dihedral potential is described in (Guo). For dihedral angles in the helical region, the energy function is represented by a standard potential consisting of three minima, one corresponding to the trans (t) state and the other to gauche states (g+ and g−). The paper describes how the A,B,C parameters are chosen so as to balance secondary (largely driven by local interactions) and tertiary structure (driven by long−range interactions).

The following coefficients must be defined for each dihedral type via the dihedral_coeff command as in the example above, or in the data file or restart files read by the read_data or read_restart commands:

- A (energy)
- B (energy)
- C (energy)

Restrictions:

This dihedral style can only be used if LAMMPS was built with the "molecular" package (which it is by default). See the Making LAMMPS section for more info on packages.

Related commands:

dihedral_coeff

Default: none

(Guo) Guo and Thirumalai, Journal of Molecular Biology, 263, 323−43 (1996).

dihedral_style hybrid command

Syntax:

```
dihedral_style hybrid style1 style2 ...
```

- style1,style2 = list of one or more dihedral styles

Examples:

```
dihedral_style hybrid harmonic helix
dihedral_coeff 1 harmonic 6.0 1 3
dihedral_coeff 2* helix 10 10 10
```

Description:

The *hybrid* style enables the use of multiple dihedral styles in one simulation. An dihedral style is assigned to each dihedral type. For example, dihedrals in a polymer flow (of dihedral type 1) could be computed with a *harmonic* potential and dihedrals in the wall boundary (of dihedral type 2) could be computed with a *helix* potential. The assignment of dihedral type to style is made via the dihedral_coeff command or in the data file.

In the dihedral_coeff commands, the name of a dihedral style must be added after the dihedral type, with the remaining coefficients being those appropriate to that style. In the example above, the 2 dihedral_coeff commands set dihedrals of dihedral type 1 to be computed with a *harmonic* potential with coefficients 6.0, 1, 3 for K, d, n. All other dihedral types (2−N) are computed with a *helix* potential with coefficients 10, 10, 10 for A, B, C.

If dihedral coefficients are specified in the data file read via the read_data command, then the same rule applies. E.g. "harmonic" or "helix", must be added after the dihedral type, for each line in the "Dihedral Coeffs" section, e.g.

```
Dihedral Coeffs

1 harmonic 6.0 1 3
2 helix 10 10 10
...
```

If *class2* is one of the dihedral hybrid styles, the same rule holds for specifying additional AngleTorsion (and EndBondTorsion, etc) coefficients either via the input script or in the data file. I.e. *class2* must be added to each line after the dihedral type. For lines in the AngleTorsion (or EndBondTorsion, etc) section of the data file for dihedral types that are not *class2*, you must use an dihedral style of *skip* as a placeholder, e.g.

```
AngleTorsion Coeffs

1 skip
2 class2 1.0 1.0 1.0 3.0 3.0 3.0 30.0 50.0
...
```

Note that it is not necessary to use the dihedral style *skip* in the input script, since AngleTorsion (or EndBondTorsion, etc) coefficients need not be specified at all for dihedral types that are not *class2*.

A dihedral style of *none* with no additional coefficients can be used in place of a dihedral style, either in a input script dihedral_coeff command or in the data file, if you desire to turn off interactions for specific dihedral types.

Restrictions:

This dihedral style can only be used if LAMMPS was built with the "molecular" package (which it is by default). See the Making LAMMPS section for more info on packages.

Unlike other dihedral styles, the hybrid dihedral style does not store dihedral coefficient info for individual sub−styles in a binary restart files. Thus when retarting a simulation from a restart file, you need to re−specify dihedral_coeff commands.

Related commands:

dihedral_coeff

Default: none

dihedral_style multi/harmonic command

Syntax:

```
dihedral_style multi/harmonic
```

Examples:

```
dihedral_style multi/harmonic
dihedral_coeff 1 20 20 20 20 20
```

Description:

The *multi/harmonic* dihedral style uses the potential

$$E = \sum_{n=1,5} A_n \cos^{n-1}(\phi)$$

The following coefficients must be defined for each dihedral type via the dihedral_coeff command as in the example above, or in the data file or restart files read by the read_data or read_restart commands:

- A1 (energy)
- A2 (energy)
- A3 (energy)
- A4 (energy)
- A5 (energy)

Restrictions:

This dihedral style can only be used if LAMMPS was built with the "molecular" package (which it is by default). See the Making LAMMPS section for more info on packages.

Related commands:

dihedral_coeff

Default: none

dihedral_style none command

Syntax:

```
dihedral_style none
```

Examples:

```
dihedral_style none
```

Description:

Using an dihedral style of none means dihedral forces are not computed, even if quadruplets of dihedral atoms were listed in the data file read by the read_data command.

Restrictions: none

Related commands: none

Default: none

dihedral_style opls command

Syntax:

```
dihedral_style opls
```

Examples:

```
dihedral_style opls
dihedral_coeff 1 90.0 90.0 90.0 70.0
```

Description:

The *opls* dihedral style uses the potential

$$E = \frac{1}{2}K_1[1+\cos(\phi)]+\frac{1}{2}K_2[1-\cos(2\phi)]+\frac{1}{2}K_3[1+\cos(3\phi)]+\frac{1}{2}K_4[1-\cos(4\phi)]$$

Note that the usual 1/2 factor is not included in the K values.

This dihedral potential is used in the OPLS force field and is described in (Watkins).

The following coefficients must be defined for each dihedral type via the dihedral_coeff command as in the example above, or in the data file or restart files read by the read_data or read_restart commands:

- K1 (energy)
- K2 (energy)
- K3 (energy)
- K4 (energy)

Restrictions:

This dihedral style can only be used if LAMMPS was built with the "molecular" package (which it is by default). See the Making LAMMPS section for more info on packages.

Related commands:

dihedral_coeff

Default: none

(Watkins) Watkins and Jorgensen, J Phys Chem A, 105, 4118−4125 (2001).

dihedral_style command

Syntax:

```
dihedral_style style
```

- style = *none* or *hybrid* or *charmm* or *class2* or *harmonic* or *helix* or *multi/harmonic* or *opls*

Examples:

```
dihedral_style harmonic
dihedral_style multi/harmonic
dihedral_style hybrid harmonic charmm
```

Description:

Set the formula(s) LAMMPS uses to compute dihedral interactions between quadruplets of atoms, which remain in force for the duration of the simulation. The list of dihedral quadruplets is read in by a read_data or read_restart command from a data or restart file.

Hybrid models where dihedrals are computed using different dihedral potentials can be setup using the *hybrid* dihedral style.

The coefficients associated with a dihedral style can be specified in a data or restart file or via the dihedral_coeff command.

All dihedral potentials store their coefficient data in binary restart files which means dihedral_style and dihedral_coeff commands do not need to be re−specified in an input script that restarts a simulation. See the read_restart command for details on how to do this. The one exception is that dihedral_style *hybrid* only stores the list of sub−styles in the restart file; dihedral coefficients need to be re−specified.

IMPORTANT NOTE: When both a dihedral and pair style is defined, the special_bonds command often needs to be used to turn off (or weight) the pairwise interaction that would otherwise exist between 4 bonded atoms.

In the formulas listed for each dihedral style, *phi* is the torsional angle defined by the quadruplet of atoms.

Here are some important points to take note of when defining the LAMMPS dihedral coefficients in the formulas that follow so that they are compatible with other force fields:

- The LAMMPS convention is that the trans position = 180 degrees, while in some force fields trans = 0 degrees.
- Some force fields reverse the sign convention on *d*.
- Some force fields divide/multiply *K* by the number of multiple torsions that contain the j−k bond in an i−j−k−l torsion.
- Some force fields let *n* be positive or negative which corresponds to *d* = 1 or −1 for the harmonic style.

Here is an alphabetic list of dihedral styles defined in LAMMPS. Click on the style to display the formula it computes and coefficients specified by the associated dihedral_coeff command:

- dihedral_style none − turn off dihedral interactions
- dihedral_style hybrid − define multiple styles of dihedral interactions

- dihedral_style charmm − CHARMM dihedral
- dihedral_style class2 − COMPASS (class 2) dihedral
- dihedral_style harmonic − harmonic dihedral
- dihedral_style helix − helix dihedral
- dihedral_style multi/harmonic − multi−harmonic dihedral
- dihedral_style opls − OPLS dihedral

There are also additional dihedral styles submitted by users which are included in the LAMMPS distribution. The list of these with links to the individual styles are given in the dihedral section of this page.

Restrictions:

Dihedral styles can only be set for atom styles that allow dihedrals to be defined.

Most dihedral styles are part of the "molecular" package. They are only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info on packages. The doc pages for individual dihedral potentials tell if it is part of a package.

Related commands:

dihedral_coeff

Default:

dihedral_style none

dimension command

Syntax:

```
dimension N
```

- N = 2 or 3

Examples:

```
dimension 2
```

Description:

Set the dimensionality of the simulation. By default LAMMPS runs 3d simulations. To run a 2d simulation, this command should be used prior to setting up a simulation box via the create_box or read_data commands. Restart files also store this setting.

See the discussion in this section for additional instructions on how to run 2d simulations.

IMPORTANT NOTE: Some models in LAMMPS treat particles as extended spheres or ellipsoids, as opposed to point particles. In 2d, the particles will still be spheres or ellipsoids, not circular disks or ellipses, meaning their moment of inertia will be the same as in 3d.

Restrictions:

This command must be used before the simulation box is defined by a read_data or create_box command.

Related commands:

fix enforce2d

Default:

```
dimension 3
```

displace_atoms command

Syntax:

```
displace_atoms group-ID style args keyword value ...
```

- group-ID = ID of group of atoms to displace
- style = *move* or *ramp* or *random*

```
  move args = delx dely delz
      delx,dely,delz = distance to displace in each dimension (distance units)
    ramp args = ddim dlo dhi dim clo chi
      ddim = x or y or z
      dlo,dhi = displacement distance between dlo and dhi (distance units)
      dim = x or y or z
      clo,chi = lower and upper bound of domain to displace (distance units)
    random args = dx dy dz seed
      dx,dy,dz = random displacement magnitude in each dimension (distance units)
      seed = random # seed (positive integer)
```

- zero or more keyword/value pairs may be appended

```
  keyword = units
    value = box or lattice
```

Examples:

```
displace_atoms top move 0 -5 0 units box
displace_atoms flow ramp x 0.0 5.0 y 2.0 20.5
```

Description:

Displace a group of atoms. This can be used to move atoms a large distance before beginning a simulation or to randomize atoms initially on a lattice. For example, in a shear simulation, an initial strain can be imposed on the system. Or two groups of atoms can be brought into closer proximity.

The *move* style displaces the group of atoms by the specified 3d distance.

The *ramp* style displaces atoms a variable amount in one dimension depending on the atom's coordinate in a (possibly) different dimension. For example, the second example command displaces atoms in the x-direction an amount between 0.0 and 5.0 distance units. Each atom's displacement depends on the fractional distance its y coordinate is between 2.0 and 20.5. Atoms with y-coordinates outside those bounds will be moved the minimum (0.0) or maximum (5.0) amount.

The *random* style independently moves each atom in the group by a random displacement, uniformly sampled from a value between -dx and +dx in the x dimension, and similarly for y and z. Random numbers are used in such a way that the displacement of a particular atom is the same, regardless of how many processors are being used.

Distance units for displacement are determined by the setting of *box* or *lattice* for the *units* keyword. *Box* means distance units as defined by the units command - e.g. Angstroms for *real* units. *Lattice* means distance units are in lattice spacings. The lattice command must have been previously used to define the lattice spacing.

Care should be taken not to move atoms on top of other atoms. After the move, atoms are remapped into the periodic simulation box if needed.

Atoms can be moved arbitrarily long distances by this command. If the simulation box is non−periodic, this can change its size or shape. This is not a problem, except that the mapping of processors to the simulation box is not changed by this command from its initial 3d configuration; see the processors command. Thus, if the box size or shape changes dramatically, the simulation may not be as well load−balanced (atoms per processor) as the initial mapping tried to achieve.

Restrictions:

This command requires inter−processor communication to migrate atoms once they have been displaced. This means that your system must be ready to perform a simulation before using this command (force fields are setup, atom masses are set, etc).

Related commands:

lattice

Default:

The option defaults are units = lattice.

displace_box command

Syntax:

```
displace_box group-ID parameter args ... keyword value ...
```

- group-ID = ID of group of atoms to displace
- one or more parameter/arg pairs may be appended

```
parameter = x or y or z or xy or xz or yz
  x, y, z args = style value(s)
    style = final or delta or scale or volume
      final values = lo hi
        lo hi = box boundaries at end of run (distance units)
      delta values = dlo dhi
        dlo dhi = change in box boundaries at end of run (distance units)
      scale values = factor
        factor = multiplicative factor for change in box length at end of run
      volume value = none = adjust this dim to preserve volume of system
  xy, xz, yz args = style value
    style = final or delta
      final value = tilt
        tilt = tilt factor at end of run (distance units)
      delta value = dtilt
        dtilt = change in tilt factor at end of run (distance units)
```

- zero or more keyword/value pairs may be appended
- keyword = *remap* or *units*

```
remap value = x or none
    x = remap coords of atoms in group into deforming box
    none = no remapping of coords
  units value = lattice or box
    lattice = distances are defined in lattice units
    box = distances are defined in simulation box units
```

Examples:

```
displace_box all xy final -2.0 z final 0.0 5.0 units box
displace_box all x scale 1.1 y volume z volume
```

Description:

Change the volume and/or shape of the simulation box. Orthogonal simulation boxes have 3 adjustable parameters (x,y,z). Triclinic (non-orthogonal) simulation boxes have 6 adjustable parameters (x,y,z,xy,xz,yz). Any or all of them can be adjusted independently and simultaneously by this command. This fix can be used to expand or contract a box, or to apply a shear strain to a non-orthogonal box.

Any parameter varied by this command must refer to a periodic dimension − see the boundary command. For parameters "xy", "xz", and "yz" this means both affected dimensions must be periodic, e.g. x and y for "xy". Dimensions not varied by this command can be periodic or non-periodic.

The size and shape of the initial simulation box are specified by the create_box or read_data or read_restart command used to setup the simulation, or they are the values from the end of the previous run. The create_box, read data, and read_restart commands also determine whether the simulation box is orthogonal or triclinic and their doc pages explain the meaning of the xy,xz,yz tilt factors. If the displace_box command changes the

xy,xz,yz tilt factors, then the simulation box must be triclinic, even if its initial tilt factors are 0.0.

For the *x*, *y*, and *z* parameters, this is the meaning of their styles and values.

For style *final*, the final lo and hi box boundaries of a dimension are specified. The values can be in lattice or box distance units. See the discussion of the units keyword below.

For style *delta*, plus or minus changes in the lo/hi box boundaries of a dimension are specified. The values can be in lattice or box distance units. See the discussion of the units keyword below.

For style *scale*, a multiplicative factor to apply to the box length of a dimension is specified. For example, if the initial box length is 10, and the factor is 1.1, then the final box length will be 11. A factor less than 1.0 means compression.

The *volume* style changes the specified dimension in such a way that the box volume remains constant while other box dimensions are changed explicitly via the styles discussed above. For example, "x scale 1.1 y scale 1.1 z volume" will shrink the z box length as the x,y box lengths increase, to keep the volume constant (product of x,y,z lengths). If "x scale 1.1 z volume" is specified and parameter *y* is unspecified, then the z box length will shrink as x increases to keep the product of x,z lengths constant. If "x scale 1.1 y volume z volume" is specified, then both the y,z box lengths will shrink as x increases to keep the volume constant (product of x,y,z lengths). In this case, the y,z box lengths shrink so as to keep their relative aspect ratio constant.

For solids or liquids, note that when one dimension of the box is expanded by this command, it may be physically undesirable to hold the other 2 box lengths constant (unspecified by this command) since that implies a density change. Using the *volume* style for those 2 dimensions to keep the box volume constant may make more physical sense, but may also not be correct for materials and potentials whose Poisson ratio is not 0.5.

For the *scale* and *volume* styles, the box length is expanded or compressed around its mid point.

For the *xy*, *xz*, and *yz* parameters, this is the meaning of their styles and values. Note that changing the tilt factors of a triclinic box does not change its volume.

For style *final*, the final tilt factor is specified. The value can be in lattice or box distance units. See the discussion of the units keyword below.

For style *delta*, a plus or minus change in the tilt factor is specified. The value can be in lattice or box distance units. See the discussion of the units keyword below.

All of these styles change the xy, xz, yz tilt factors. In LAMMPS, tilt factors (xy,xz,yz) for triclinic boxes are always bounded by half the distance of the parallel box length. For example, if xlo = 2 and xhi = 12, then the x box length is 10 and the xy tilt factor must be between −5 and 5. Similarly, both xz and yz must be between −(xhi−xlo)/2 and +(yhi−ylo)/2. Note that this is not a limitation, since if the maximum tilt factor is 5 (as in this example), then configurations with tilt = ..., −15, −5, 5, 15, 25, ... are all equivalent. Any tilt factor specified by this command must be within these limits.

The *remap* keyword determines whether atom positions are re−mapped to the new box. If *remap* is set to *x* (the default), atoms in the fix group are re−mapped; otherwise they are not. If *remap* is set to *none*, then this remapping does not take place.

The *units* keyword determines the meaning of the distance units used to define various arguments. A *box* value selects standard distance units as defined by the units command, e.g. Angstroms for units = real or metal. A *lattice* value means the distance units are in lattice spacings. The lattice command must have been previously used to

define the lattice spacing.

The simulation box size or shape can be changed by arbitrarily large amounts by this command. This is not a problem, except that the mapping of processors to the simulation box is not changed by this command from its initial 3d configuration; see the processors command. Thus, if the box size or shape changes dramatically, the simulation may not be as well load−balanced (atoms per processor) as the initial mapping tried to achieve.

Restrictions:

Any box dimension varied by this fix must be periodic.

This command requires inter−processor communication to migrate atoms once they have moved. This means that your system must be ready to perform a simulation before using this command (force fields are setup, atom masses are set, etc).

Related commands:

fix deform

Default:

The option defaults are remap = x and units = lattice.

dump command

dump image command

Syntax:

```
dump ID group-ID style N file args
```

- ID = user−assigned name for the dump
- group−ID = ID of the group of atoms to be dumped
- style = *atom* or *cfg* or *dcd* or *xtc* or *xyz* or *image* or *local* or *custom*
- N = dump every this many timesteps
- file = name of file to write dump info to
- args = list of arguments for a particular style

```
atom args = none
  cfg args = same as custom args, see below
  dcd args = none
  xtc args = none
  xyz args = none

image args = discussed on dump image doc page

local args = list of local attributes
    possible attributes = index, c_ID, c_ID[N], f_ID, f_ID[N]
      index = enumeration of local values
      c_ID = local vector calculated by a compute with ID
      c_ID[N] = Nth column of local array calculated by a compute with ID
      f_ID = local vector calculated by a fix with ID
      f_ID[N] = Nth column of local array calculated by a fix with ID

custom args = list of atom attributes
    possible attributes = id, mol, type, element, mass,
                          x, y, z, xs, ys, zs, xu, yu, zu,
                          xsu, ysu, zsu, ix, iy, iz,
                          vx, vy, vz, fx, fy, fz,
                          q, mux, muy, muz, mu,
                          radius, diameter, omegax, omegay, omegaz,
                          angmomx, angmomy, angmomz, tqx, tqy, tqz,
                          spin, eradius, ervel, erforce,
                          c_ID, c_ID[N], f_ID, f_ID[N], v_name

      id = atom ID
      mol = molecule ID
      type = atom type
      element = name of atom element, as defined by dump_modify command
      mass = atom mass
      x,y,z = unscaled atom coordinates
      xs,ys,zs = scaled atom coordinates
      xu,yu,zu = unwrapped atom coordinates
      xsu,ysu,zsu = scaled unwrapped atom coordinates
      ix,iy,iz = box image that the atom is in
      vx,vy,vz = atom velocities
      fx,fy,fz = forces on atoms
      q = atom charge
      mux,muy,muz = orientation of dipole moment of atom
      mu = magnitude of dipole moment of atom
```

```
radius,diameter = radius,diameter of spherical particle
omegax,omegay,omegaz = angular velocity of extended particle
angmomx,angmomy,angmomz = angular momentum of extended particle
tqx,tqy,tqz = torque on extended particles
spin = electron spin
eradius = electron radius
ervel = electron radial velocity
erforce = electron radial force
c_ID = per-atom vector calculated by a compute with ID
c_ID[N] = Nth column of per-atom array calculated by a compute with ID
f_ID = per-atom vector calculated by a fix with ID
f_ID[N] = Nth column of per-atom array calculated by a fix with ID
v_name = per-atom vector calculated by an atom-style variable with name
```

Examples:

```
dump myDump all atom 100 dump.atom
dump 2 subgroup atom 50 dump.run.bin
dump 4a all custom 100 dump.myforce.* id type x y vx fx
dump 4b flow custom 100 dump.%.myforce id type c_myF[3] v_ke
dump 2 inner cfg 10 dump.snap.*.cfg id type xs ys zs vx vy vz
dump snap all cfg 100 dump.config.*.cfg id type xs ys zs id type c_Stress2
dump 1 all xtc 1000 file.xtc
dump e_data all custom 100 dump.eff id type x y z spin eradius fx fy fz eforce
```

Description:

Dump a snapshot of atom quantities to one or more files every N timesteps in one of several styles. The *image* style is the exception; it creates a JPG or PPM image file of the atom configuration every N timesteps, as discussed on the dump image doc page.

Only information for atoms in the specified group is dumped. The dump_modify thresh and region commands can also alter what atoms are included. Not all styles support all these options; see details below.

As described below, the filename determines the kind of output (text or binary or gzipped, one big file or one per timestep, one big file or one per processor).

IMPORTANT NOTE: Because periodic boundary conditions are enforced only on timesteps when neighbor lists are rebuilt, the coordinates of an atom written to a dump file may be slightly outside the simulation box.

IMPORTANT NOTE: Unless the dump_modify sort option is invoked, the lines of atom information written to dump files (typically one line per atom) will be in an indeterminate order for each snapshot. This is even true when running on a single processor, if the atom_modify sort option is on, which it is by default. In this case atoms are re-ordered periodically during a simulation, due to spatial sorting. It is also true when running in parallel, because data for a single snapshot is collected from multiple processors.

For the *atom*, *custom*, *cfg*, and *local* styles, sorting is off by default. For the *dcd*, *xtc*, and *xyz* styles, sorting by atom ID is on by default. See the dump_modify doc page for details.

The *style* keyword determines what atom quantities are written to the file and in what format. Settings made via the dump_modify command can also alter the format of individual values and the file itself.

The *atom*, *local*, and *custom* styles create files in a simple text format that is self-explanatory when viewing a dump file. Many of the LAMMPS post-processing tools, including Pizza.py, work with this format.

For post-processing purposes the *atom* and *custom* text files are self-describing in the following sense.

The dimensions of the simulation box are included in each snapshot. For an orthogonal simulation box this information is is formatted as:

```
ITEM: BOX BOUNDS xx yy zz
xlo xhi
ylo yhi
zlo zhi
```

where xlo,xhi are the maximum extents of the simulation box in the x−dimension, and similarly for y and z. The "xx yy zz" represent 6 characters that encode the style of boundary for each of the 6 simulation box boundaries (xlo,xhi and ylo,yhi and zlo,zhi). Each of the 6 characters is either p = periodic, f = fixed, s = shrink wrap, or m = shrink wrapped with a minimum value. See the boundary command for details.

For triclinic simulation boxes (non−orthogonal), an orthogonal bounding box which encloses the triclinic simulation box is output, along with the 3 tilt factors (xy, xz, yz) of the triclinic box, formatted as follows:

```
ITEM: BOX BOUNDS xy xz yz xx yy zz
xlo_bound xhi_bound xy
ylo_bound yhi_bound xz
zlo_bound zhi_bound yz
```

The presence of the text "xy xz yz" in the ITEM line indicates that the 3 tilt factors will be included on each of the 3 following lines. This bounding box is convenient for many visualization programs. The meaning of the 6 character flags for "xx yy zz" is the same as above.

Note that the first two numbers on each line are now xlo_bound instead of xlo, etc, since they repesent a bounding box. See this section of the doc pages for a geometric description of triclinic boxes, as defined by LAMMPS, simple formulas for how the 6 bounding box extents (xlo_bound,xhi_bound,etc) are calculated from the triclinic parameters, and how to transform those parameters to and from other commonly used triclinic representations.

The "ITEM: ATOMS" line in each snapshot lists column descriptors for the per−atom lines that follow. For example, the descriptors would be "id type xs ys zs" for the default *atom* style, and would be the atom attributes you specify in the dump command for the *custom* style.

For style *atom*, atom coordinates are written to the file, along with the atom ID and atom type. By default, atom coords are written in a scaled format (from 0 to 1). I.e. an x value of 0.25 means the atom is at a location 1/4 of the distance from xlo to xhi of the box boundaries. The format can be changed to unscaled coords via the dump_modify settings. Image flags can also be added for each atom via dump_modify.

Style *custom* allows you to specify a list of atom attributes to be written to the dump file for each atom. Possible attributes are listed above and will appear in the order specified. You cannot specify a quantity that is not defined for a particular simulation − such as *q* for atom style *bond*, since that atom style doesn't assign charges. Dumps occur at the very end of a timestep, so atom attributes will include effects due to fixes that are applied during the timestep. An explanation of the possible dump custom attributes is given below.

For style *local*, local output generated by computes and fixes is used to gnerate lines of output that is written to the dump file. This local data is typically calculated by each processor based on the atoms it owns, but there may be zero or more entities per atom, e.g. a list of bond distances. An explanation of the possible dump local attributes is given below. Note that by using input from the compute property/local command with dump local, it is possible to generate information on bonds, angles, etc that can be cut and pasted directly into a data file read by the read_data command.

Style *cfg* has the same command syntax as style *custom* and writes extended CFG format files, as used by the AtomEye visualization package. Since the extended CFG format uses a single snapshot of the system per file, a

wildcard "*" must be included in the filename, as discussed below. The list of atom attributes for style *cfg* must begin with either "id type xs ys zs" or "id type xsu ysu zsu" or since these quantities are needed to write the CFG files in the appropriate format (though the "id" and "type" fields do not appear explicitly in the file). Any remaining attributes will be stored as "auxiliary properties" in the CFG files. Note that you will typically want to use the dump_modify element command with CFG−formatted files, to associate element names with atom types, so that AtomEye can render atoms appropriately. When unwrapped coordinates *xsu*, *ysu*, and *zsu* are requested, the nominal AtomEye periodic cell dimensions are expanded by a large factor UNWRAPEXPAND = 10.0, which ensures atoms that are displayed correctly for up to UNWRAPEXPAND/2 periodic boundary crossings in any direction. Beyond this, AtomEye will rewrap the unwrapped coordinates. The expansion causes the atoms to be drawn farther away from the viewer, but it is easy to zoom the atoms closer, and the interatomic distances are unaffected.

The *dcd* style writes DCD files, a standard atomic trajectory format used by the CHARMM, NAMD, and XPlor molecular dynamics packages. DCD files are binary and thus may not be portable to different machines. The number of atoms per snapshot cannot change with the *dcd* style. The *unwrap* option of the dump_modify command allows DCD coordinates to be written "unwrapped" by the image flags for each atom. Unwrapped means that if the atom has passed through a periodic boundary one or more times, the value is printed for what the coordinate would be if it had not been wrapped back into the periodic box. Note that these coordinates may thus be far outside the box size stored with the snapshot.

The *xtc* style writes XTC files, a compressed trajectory format used by the GROMACS molecular dynamics package, and described here. The precision used in XTC files can be adjusted via the dump_modify command. The default value of 1000 means that coordinates are stored to 1/1000 nanometer accuracy. XTC files are portable binary files written in the NFS XDR data format, so that any machine which supports XDR should be able to read them. The number of atoms per snapshot cannot change with the *xtc* style. The *unwrap* option of the dump_modify command allows XTC coordinates to be written "unwrapped" by the image flags for each atom. Unwrapped means that if the atom has passed thru a periodic boundary one or more times, the value is printed for what the coordinate would be if it had not been wrapped back into the periodic box. Note that these coordinates may thus be far outside the box size stored with the snapshot.

The *xyz* style writes XYZ files, which is a simple text−based coordinate format that many codes can read.

Note that DCD, XTC, and XYZ formatted files can be read directly by VMD (a popular molecular viewing program). See this section of the manual and the tools/lmp2vmd/README.txt file for more information about support in VMD for reading and visualizing LAMMPS dump files.

Dumps are performed on timesteps that are a multiple of N (including timestep 0) and on the last timestep of a minimization if the minimization converges. Note that this means a dump will not be performed on the initial timestep after the dump command is invoked, if the current timestep is not a multiple of N. This behavior can be changed via the dump_modify first command, which can be useful if the dump command is invoked after a minimization ended on an arbitrary timestep. N can be changed between runs by using the dump_modify every command (not allowed for *dcd* style).

The specified filename determines how the dump file(s) is written. The default is to write one large text file, which is opened when the dump command is invoked and closed when an undump command is used or when LAMMPS exits. For the *dcd* and *xtc* styles, this is a single large binary file.

Dump filenames can contain two wildcard characters. If a "*" character appears in the filename, then one file per snapshot is written and the "*" character is replaced with the timestep value. For example, tmp.dump.* becomes tmp.dump.0, tmp.dump.10000, tmp.dump.20000, etc. This option is not available for the *dcd* and *xtc* styles. Note that the dump_modify pad command can be used to insure all timestep numbers are the same length (e.g. 00010), which can make it easier to read a series of dump files in order by some post−processing tools.

If a "%" character appears in the filename, then one file is written for each processor and the "%" character is replaced with the processor ID from 0 to P−1. For example, tmp.dump.% becomes tmp.dump.0, tmp.dump.1, ... tmp.dump.P−1, etc. This creates smaller files and can be a fast mode of output on parallel machines that support parallel I/O for output. This option is not available for the *dcd*, *xtc*, and *xyz* styles.

Note that the "*" and "%" characters can be used together to produce a large number of small dump files!

If the filename ends with ".bin", the dump file (or files, if "*" or "%" is also used) is written in binary format. A binary dump file will be about the same size as a text version, but will typically write out much faster. Of course, when post−processing, you will need to convert it back to text format (see the binary2txt tool) or write your own code to read the binary file. The format of the binary file can be understood by looking at the tools/binary2txt.cpp file. This option is only available for the *atom* and *custom* styles.

If the filename ends with ".gz", the dump file (or files, if "*" or "%" is also used) is written in gzipped format. A gzipped dump file will be about 3x smaller than the text version, but will also take longer to write. This option is not available for the *dcd* and *xtc* styles.

This section explains the local attributes that can be specified as part of the *local* style.

The *index* attribute can be used to generate an index number from 1 to N for each line written into the dump file, where N is the total number of local datums from all processors, or lines of output that will appear in the snapshot. Note that because data from different processors depend on what atoms they currently own, and atoms migrate between processor, there is no guarantee that the same index will be used for the same info (e.g. a particular bond) in successive snapshots.

The *c_ID* and *c_ID[N]* attributes allow local vectors or arrays calculated by a compute to be output. The ID in the attribute should be replaced by the actual ID of the compute that has been defined previously in the input script. See the compute command for details. There are computes for calculating local information such as indices, types, and energies for bonds and angles.

Note that computes which calculate global or per−atom quantities, as opposed to local quantities, cannot be output in a dump local command. Instead, global quantities can be output by the thermo_style custom command, and per−atom quantities can be output by the dump custom command.

If *c_ID* is used as a attribute, then the local vector calculated by the compute is printed. If *c_ID[N]* is used, then N must be in the range from 1−M, which will print the Nth column of the M−length local array calculated by the compute.

The *f_ID* and *f_ID[N]* attributes allow local vectors or arrays calculated by a fix to be output. The ID in the attribute should be replaced by the actual ID of the fix that has been defined previously in the input script.

If *f_ID* is used as a attribute, then the local vector calculated by the fix is printed. If *f_ID[N]* is used, then N must be in the range from 1−M, which will print the Nth column of the M−length local array calculated by the fix.

This section explains the atom attributes that can be specified as part of the *custom* and *cfg* styles.

The *id*, *mol*, *type*, *element*, *mass*, *vx*, *vy*, *vz*, *fx*, *fy*, *fz*, *q* attributes are self−explanatory.

Id is the atom ID. *Mol* is the molecule ID, included in the data file for molecular systems. *Type* is the atom type. *Element* is typically the chemical name of an element, which you must assign to each type via the dump_modify element command. More generally, it can be any string you wish to associated with an atom type. *Mass* is the atom mass. *Vx*, *vy*, *vz*, *fx*, *fy*, *fz*, and *q* are components of atom velocity and force and atomic charge.

There are several options for outputting atom coordinates. The *x*, *y*, *z* attributes write atom coordinates "unscaled", in the appropriate distance units (Angstroms, sigma, etc). Use *xs*, *ys*, *zs* if you want the coordinates "scaled" to the box size, so that each value is 0.0 to 1.0. If the simulation box is triclinic (tilted), then all atom coords will still be between 0.0 and 1.0. Use *xu*, *yu*, *zu* if you want the coordinates "unwrapped" by the image flags for each atom. Unwrapped means that if the atom has passed thru a periodic boundary one or more times, the value is printed for what the coordinate would be if it had not been wrapped back into the periodic box. Note that using *xu*, *yu*, *zu* means that the coordinate values may be far outside the box bounds printed with the snapshot. Using *xsu*, *ysu*, *zsu* is similar to using *xu*, *yu*, *zu*, except that the unwrapped coordinates are scaled by the box size. Atoms that have passed through a periodic boundary will have the corresponding cooordinate increased or decreased by 1.0.

The image flags can be printed directly using the *ix*, *iy*, *iz* attributes. The dump_modify command describes in more detail what is meant by scaled vs unscaled coordinates and the image flags.

The *mux*, *muy*, *muz* attributes are specific to dipolar systems defined with an atom style of *dipole*. They give the orientation of the atom's point dipole moment. The *mu* attribute gives the magnitude of the atom's dipole moment.

The *radius* and *diameter* attributes are specific to extended spherical particles that have a finite size, such as those defined with an atom style of *sphere*.

The *omegax*, *omegay*, and *omegaz* attributes are specific to extended spherical or aspherical particles that have an angular velocity. Only certain atom styles, such as *sphere* define this quantity.

The *angmomx*, *angmomy*, and *angmomz* attributes are specific to extended aspherical particles that have an angular momentum. Only the *ellipsoid* atom style defines this quantity.

The *tqx*, *tqy*, *tqz* attributes are for extended spherical or aspherical particles that can sustain a rotational torque due to interactions with other particles.

The *spin*, *eradius*, *ervel*, and *erforce* attributes are for particles that represent nuclei and electrons modeled with the electronic force field (EFF). See atom_style electron and pair_style eff for more details.

The *c_ID* and *c_ID[N]* attributes allow per−atom vectors or arrays calculated by a compute to be output. The ID in the attribute should be replaced by the actual ID of the compute that has been defined previously in the input script. See the compute command for details. There are computes for calculating the per−atom energy, stress, centro−symmetry parameter, and coordination number of individual atoms.

Note that computes which calculate global or local quantities, as opposed to per−atom quantities, cannot be output in a dump custom command. Instead, global quantities can be output by the thermo_style custom command, and local quantities can be output by the dump local command.

If *c_ID* is used as a attribute, then the per−atom vector calculated by the compute is printed. If *c_ID[N]* is used, then N must be in the range from 1−M, which will print the Nth column of the M−length per−atom array calculated by the compute.

The *f_ID* and *f_ID[N]* attributes allow vector or array per−atom quantities calculated by a fix to be output. The ID in the attribute should be replaced by the actual ID of the fix that has been defined previously in the input script. The fix ave/atom command is one that calculates per−atom quantities. Since it can time−average per−atom quantities produced by any compute, fix, or atom−style variable, this allows those time−averaged results to be written to a dump file.

If *f_ID* is used as a attribute, then the per−atom vector calculated by the fix is printed. If *f_ID[N]* is used, then N must be in the range from 1−M, which will print the Nth column of the M−length per−atom array calculated by

the fix.

The *v_name* attribute allows per−atom vectors calculated by a variable to be output. The name in the attribute should be replaced by the actual name of the variable that has been defined previously in the input script. Only an atom−style variable can be referenced, since it is the only style that generates per−atom values. Variables of style *atom* can reference individual atom attributes, per−atom atom attributes, thermodynamic keywords, or invoke other computes, fixes, or variables when they are evaluated, so this is a very general means of creating quantities to output to a dump file.

See this section of the manual for information on how to add new compute and fix styles to LAMMPS to calculate per−atom quantities which could then be output into dump files.

Restrictions:

To write gzipped dump files, you must compile LAMMPS with the −DLAMMPS_GZIP option − see the Making LAMMPS section of the documentation.

The *xtc* style is part of the "xtc" package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info. This is because some machines may not support the low−level XDR data format that XTC files are written with, which will result in a compile−time error when a low−level include file is not found. Putting this style in a package makes it easy to exclude from a LAMMPS build for those machines. However, the XTC package also includes two compatibility header files and associated functions, which should be a suitable substitute on machines that do not have the appropriate native header files. This option can be invoked at build time by adding −DLAMMPS_XDR to the CCFLAGS variable in the appropriate low−level Makefile, e.g. src/MAKE/Makefile.foo. This compatibility mode has been tested successfully on Cray XT3/XT4/XT5 and IBM BlueGene/L machines and should also work on IBM BG/P, and Windows XP/Vista/7 machines.

Related commands:

dump image, dump_modify, undump

Default:

The defaults for the image style are listed on the dump image doc page.

dump image command

Syntax:

```
dump ID group-ID image N file color diameter keyword value ...
```

- ID = user−assigned name for the dump
- group−ID = ID of the group of atoms to be imaged
- image = style of dump command (other styles *atom* or *cfg* or *dcd* or *xtc* or *xyz* or *local* or *custom* are discussed on the dump doc page)
- N = dump every this many timesteps
- file = name of file to write image to
- color = atom attribute that determines color of each atom
- diameter = atom attribute that determines size of each atom
- zero or more keyword/value pairs may be appended
- keyword = *adiam* or *atom* or *bond* or *size* or *view* or *center* or *up* or *zoom* or *persp* or *box* or *axes* or *shiny* or *ssao*

```
adiam value = number = numeric value for atom diameter (distance units)
  atom = yes/no = do or do not draw atoms
  bond values = color width = color and width of bonds
    color = atom or type or none
    width = number or atom or type or none
      number = numeric value for bond width (distance units)
  size values = width height = size of images
    width = width of image in # of pixels
    height = height of image in # of pixels
  view values = theta phi = view of simulation box
    theta = view angle from +z axis (degrees)
    phi = azimuthal view angle (degrees)
    theta or phi can be a variable (see below)
  center values = flag Cx Cy Cz = center point of image
    flag = "s" for static, "d" for dynamic
    Cx,Cy,Cz = center point of image as fraction of box dimension (0.5 = center of box)
    Cx,Cy,Cz can be variables (see below)
  up values = Ux Uy Uz = direction that is "up" in image
    Ux,Uy,Uz = components of up vector
    Ux,Uy,Uz can be variables (see below)
  zoom value = zfactor = size that simulation box appears in image
    zfactor = scale image size by factor > 1 to enlarge, factor <1 to shrink
    zfactor can be a variable (see below)
  persp value = pfactor = amount of "perspective" in image
    pfactor = amount of perspective (0 = none,  <1 = some, > 1 = highly skewed)
    pfactor can be a variable (see below)
  box values = yes/no diam = draw outline of simulation box
    yes/no = do or do not draw simulation box lines
    diam = diameter of box lines as fraction of shortest box length
  axes values = yes/no length diam = draw xyz axes
    yes/no = do or do not draw xyz axes lines next to simulation box
    length = length of axes lines as fraction of respective box lengths
    diam = diameter of axes lines as fraction of shortest box length
  shiny value = sfactor = shinyness of spheres and cylinders
    sfactor = shinyness of spheres and cylinders from 0.0 to 1.0
  ssao value = yes/no seed dfactor = SSAO depth shading
    yes/no = turn depth shading on/off
    seed = random # seed (positive integer)
    dfactor = strength of shading from 0.0 to 1.0
```

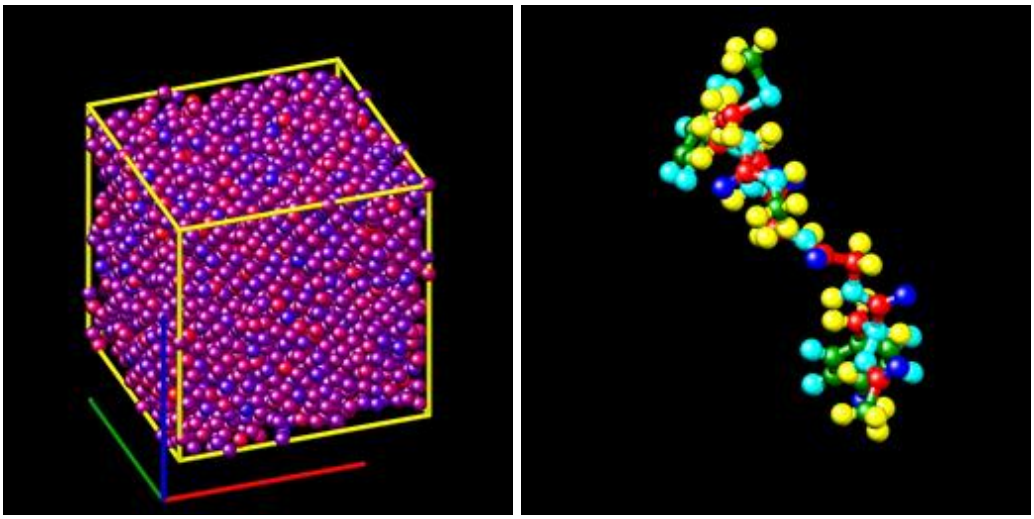
Examples:

```
dump myDump all image 100 dump.*.jpg type type
```

Description:

Dump a high−quality ray−traced image of the atom configuration every N timesteps as either a JPG or PPM file. A series of such images can easily be converted into an animated movie of your simulation; see further details below. Other dump styles store snapshots of numerical data asociated with atoms in various formats, as discussed on the dump doc page.

Here are two sample images, rendered as 1024x1024 JPG files. Click to see the full−size images:



Only atoms in the specified group are rendered in the image. The dump_modify region and thresh commands can also alter what atoms are included in the image.

The filename suffix determines whether a JPG or PPM file is created. If the suffix is ".jpg" or ".jpeg", then a JPG file is created, else a PPM file is created, which is a text−based format. To write out JPG files, you must build LAMMPS with a JPEG library. See this section of the manual for instructions on how to do this.

IMPORTANT NOTE: Because periodic boundary conditions are enforced only on timesteps when neighbor lists are rebuilt, the coordinates of an atom in the image may be slightly outside the simulation box.

Dumps are performed on timesteps that are a multiple of N (including timestep 0) and on the last timestep of a minimization if the minimization converges. Note that this means a dump will not be performed on the initial timestep after the dump command is invoked, if the current timestep is not a multiple of N. This behavior can be changed via the dump_modify first command, which can be useful if the dump command is invoked after a minimization ended on an arbitrary timestep. N can be changed between runs by using the dump_modify every command.

Dump image filenames must contain a wildcard character "*", so that one image file per snapshot is written. The "*" character is replaced with the timestep value. For example, tmp.dump.*.jpg becomes tmp.dump.0.jpg, tmp.dump.10000.jpg, tmp.dump.20000.jpg, etc. Note that the dump_modify pad command can be used to insure all timestep numbers are the same length (e.g. 00010), which can make it easier to convert a series of images into a movie in the correct ordering.

The *color* and *diameter* settings determine the color and size of atoms rendered in the image. They can be any atom attribute defined for the dump custom command, including *type* and *element*. This includes per−atom

quantities calculated by a compute, fix, or variable, which are prefixed by "c_", "f_", or "v_" respectively. Note that the *diameter* setting can be overridden with a numeric value by the optional *adiam* keyword, in which case you can specify the *diameter* setting with any valid atom attribute.

If *type* is specified for the *color* setting, then the color of each atom is determined by its atom type. By default the mapping of types to colors is as follows:

- type 1 = red
- type 2 = green
- type 3 = blue
- type 4 = yellow
- type 5 = aqua
- type 6 = cyan

and repeats itself for types > 6. This mapping can be changed by the dump_modify acolor command.

If *type* is specified for the *diameter* setting then the diameter of each atom is determined by its atom type. By default all types have diameter 1.0. This mapping can be changed by the dump_modify adiam command.

If *element* is specified for the *color* and/or *diameter* setting, then the color and/or diameter of each atom is determined by which element it is, which in turn is specified by the element−to−type mapping specified by the "dump_modify element" command. By default every atom type is C (carbon). Every element has a color and diameter associated with it, which is the same as the colors and sizes used by the AtomEye visualization package.

If other atom attributes are used for the *color* or *diameter* settings, they are interpreted in the following way.

If "vx", for example, is used as the *color* setting, then the color of the atom will depend on the x−component of its velocity. The association of a per−atom value with a specific color is determined by a "color map", which can be specified via the dump_modify command. The basic idea is that the atom−attribute will be within a range of values, and every value within the range is mapped to a specific color. Depending on how the color map is defined, that mapping can take place via interpolation so that a value of −3.2 is halfway between "red" and "blue", or discretely so that the value of −3.2 is "orange".

If "vx", for example, is used as the *diameter* setting, then the atom will be rendered using the x−component of its velocity as the diameter. If the per−atom value <= 0.0, them the atom will not be drawn. Note that finite−size spherical particles, as defined by atom_style sphere define a per−particle radius or diameter, which can be used as the *diameter* setting.

The various kewords listed above control how the image is rendered. As listed below, all of the keywords have defaults, most of which you will likely not need to change. The dump modify also has options specific to the dump image style, particularly for assigning colors to atoms, bonds, and other image features.

The *adiam* keyword allows you to override the *diameter* setting to a per−atom attribute with a specified numeric value. All atoms will be drawn with that diameter, e.g. 1.5, which is in whatever distance units the input script defines, e.g. Angstroms.

The *atom* keyword allow you to turn off the drawing of all atoms, if the specified value is *no*.

The *bond* keyword allows to you to alter how bonds are drawn. A bond is only drawn if both atoms in the bond are being drawn due to being in the specified group and due to other selection criteria (e.g. region, threshhold settings of the dump_modify command). By default, bonds are drawn if they are defined in the input data file as read by the read_data command. Using *none* for both the bond *color* and *width* value will turn off the drawing of

all bonds.

If *atom* is specified for the bond *color* value, then each bond is drawn in 2 halves, with the color of each half being the color of the atom at that end of the bond.

If *type* is specified for the *color* value, then the color of each bond is determined by its bond type. By default the mapping of bond types to colors is as follows:

- type 1 = red
- type 2 = green
- type 3 = blue
- type 4 = yellow
- type 5 = aqua
- type 6 = cyan

and repeats itself for bond types > 6. This mapping can be changed by the dump_modify bcolor command.

The bond *width* value can be a numeric value or *atom* or *type* (or *none* as indicated above).

If a numeric value is specified, then all bonds will be drawn as cylinders with that diameter, e.g. 1.0, which is in whatever distance units the input script defines, e.g. Angstroms.

If *atom* is specified for the *width* value, then each bond will be drawn with a width corresponding to the minimum diameter of the 2 atoms in the bond.

If *type* is specified for the *width* value then the diameter of each bond is determined by its bond type. By default all types have diameter 0.5. This mapping can be changed by the dump_modify bdiam command.

The *size* keyword sets the width and height of the created images, i.e. the number of pixels in each direction.

The *view*, *center*, *up*, *zoom*, and *persp* values determine how 3d simulation space is mapped to the 2d plane of the image. Basically they control how the simulation box appears in the image.

All of the *view*, *center*, *up*, *zoom*, and *persp* values can be specified as numeric quantities, whose meaning is explained below. Any of them can also be specified as an equal−style variable, by using v_name as the value, where "name" is the variable name. In this case the variable will be evaluated on the timestep each image is created to create a new value. If the equal−style variable is time−dependent, this is a means of changing the way the simulation box appears from image to image, effectively doing a pan or fly−by view of your simulation.

The *view* keyword determines the viewpoint from which the simulation box is viewed, looking towards the *center* point. The *theta* value is the vertical angle from the +z axis, and must be an angle from 0 to 180 degrees. The *phi* value is an azimuthal angle around the z axis and can be positive or negative. A value of 0.0 is a view along the +x axis, towards the *center* point. If *theta* or *phi* are specified via variables, then the variable values should be in degrees.

The *center* keyword determines the point in simulation space that will be at the center of the image. *Cx*, *Cy*, and *Cz* are speficied as fractions of the box dimensions, so that (0.5,0.5,0.5) is the center of the simulation box. These values do not have to be between 0.0 and 1.0, if you want the simulation box to be offset from the center of the image. Note, however, that if you choose strange values for *Cx*, *Cy*, or *Cz* you may get a blank image. Internally, *Cx*, *Cy*, and *Cz* are converted into a point in simulation space. If *flag* is set to "s" for static, then this conversion is done once, at the time the dump command is issued. If *flag* is set to "d" for dynamic then the conversion is performed every time a new image is created. If the box size or shape is changing, this will adjust the center point

in simulation space.

The *up* keyword determines what direction in simulation space will be "up" in the image. Internally it is stored as a vector that is in the plane perpendicular to the view vector implied by the *theta* and *pni* values, and which is also in the plane defined by the view vector and user−specified up vector. Thus this internal vector is computed from the user−specified *up* vector as

```
up_internal = view cross (up cross view)
```

This means the only restriction on the specified *up* vector is that it cannot be parallel to the *view* vector, implied by the *theta* and *phi* values.

The *zoom* keyword scales the size of the simulation box as it appears in the image. The default *zfactor* value of 1 should display an image mostly filled by the atoms in the simulation box. A *zfactor* > 1 will make the simulation box larger; a *zfactor* < 1 will make it smaller. *Zfactor* must be a value > 0.0.

The *persp* keyword determines how much depth perspective is present in the image. Depth perspective makes lines that are parallel in simulation space appear non−parallel in the image. A *pfactor* value of 0.0 means that parallel lines will meet at infininty (1.0/pfactor), which is an orthographic rendering with no persepctive. A *pfactor* value between 0.0 and 1.0 will introduce more perspective. A *pfactor* value > 1 will create a highly skewed image with a large amount of perspective.

IMPORTANT NOTE: The *persp* keyword is not yet supported as an option.

The *box* keyword determines how the simulation box boundaries are rendered as thin cylinders in the image. If *no* is set, then the box boundaries are not drawn and the *diam* setting is ignored. If *yes* is set, the 12 edges of the box are drawn, with a diameter that is a fraction of the shortest box length in x,y,z (for 3d) or x,y (for 2d). The color of the box boundaries can be set with the dump_modify boxcolor command.

The *axes* keyword determines how the coordinate axes are rendered as thin cylinders in the image. If *no* is set, then the axes are not drawn and the *length* and *diam* settings are ignored. If *yes* is set, 3 thin cylinders are drawn to represent the x,y,z axes in colors red,green,blue. The origin of these cylinders will be offset from the lower left corner of the box by 10%. The *length* setting determines how long the cylinders will be as a fraction of the respective box lengths. The *diam* setting determines their thickness as a fraction of the shortest box length in x,y,z (for 3d) or x,y (for 2d).

The *shiny* keyword determines how shiny the objects rendered in the image will appear. The *sfactor* value must be a value 0.0 <= *sfactor* <= 1.0, where *sfactor* = 1 is a highly reflective surface and *sfactor* = 0 is a rough non−shiny surface.

The *ssao* keyword turns on/off a screen space ambient occlusion (SSAO) model for depth shading. If *yes* is set, then atoms further away from the viewer are darkened via a randomized process, which is perceived as depth. The calculation of this effect can increase the cost of computing the image by roughly 2x. The strength of the effect can be scaled by the *dfactor* parameter. If *no* is set, no depth shading is performed.

A series of JPG or PPM images can be converted into a movie file and then played as a movie using commonly available tools.

Convert JPG or PPM files into an animated GIF or MPEG or other movie file:

- a) Use the ImageMagick convert program.

```
% convert *.jpg foo.gif
% convert *.ppm foo.mpg
```

- b) Use QuickTime.

  Select "Open Image Sequence" under the File menu Load the images into QuickTime to animate them
  Select "Export" under the File menu Save the movie as a QuickTime movie (*.mov) or in another format
- c) Windows−based tool.

If someone tells us how to do this via a common Windows−based tool, we'll post the instructions here.

Play the movie:

- a) Use your browser to view an animated GIF movie.

  Select "Open File" under the File menu Load the animated GIF file
- b) Use the freely available mplayer tool to view an MPEG movie.

```
% mplayer foo.mpg
```

- c) Use the Pizza.py animate tool, which works directly on a series of image files.

```
a = animate("foo*.jpg")
```

- d) QuickTime and other Windows−based media players can obviously play movie files directly.

See this section of the manual for information on how to add new compute and fix styles to LAMMPS to calculate per−atom quantities which could then be output into dump files.

Restrictions:

To write JPG images, you must use a −DLAMMPS_JPEG switch when building LAMMPS and link with a JPEG library. See the Making LAMMPS section of the documentation for details.

Related commands:

dump, dump_modify, undump

Default:

The defaults for the keywords are as follows:

- adiam = not specified (use diameter setting)
- atom = yes
- bond = none none (if no bonds in system)
- bond = atom 0.5 (if bonds in system)
- size = 512 512
- view = 60 30 (for 3d)
- view = 0 0 (for 2d)
- center = s 0.5 0.5 0.5
- up = 0 0 1 (for 3d)
- up = 0 1 0 (for 2d)
- zoom = 1.0
- persp = 0.0
- box = yes 0.02
- axes = no 0.0 0.0

- shiny = 1.0
- ssao = no

dump_modify command

Syntax:

```
dump_modify dump-ID keyword values ...
```

- dump-ID = ID of dump to modify
- one or more keyword/value pairs may be appended
- keyword = *acolor* or *adiam* or *amap* or *append* or *bcolor* or *bdiam* or *backcolor* or *boxcolor* or *color* or *every* or *flush* or *format* or *image* or *label* or *precision* or *region* or *scale* or *sort* or *thresh* or *unwrap*

```
  acolor args = type color
      type = atom type or range of types (see below)
      color = name of color or color1/color2/...
    adiam args = type diam
      type = atom type or range of types (see below)
      diam = diameter of atoms of that type (distance units)
    amap args = lo hi style N delta entry1 entry2 ... entryN
      lo = number or min = lower bound of range of color map
      hi = number or max = upper bound of range of color map
      style = 2 letters = "c" or "d" or "s" plus "a" or "f"
        "c" for continuous
        "d" for discrete
        "s" for sequential
        "a" for absolute
        "f" for fractional
      delta = binsize (only used for style "s", otherwise ignored)
        binsize = range is divided into bins of this width
      N = # of subsequent entries
      entry = value color (for continuous style)
        value = number or min or max = single value within range
        color = name of color used for that value
      entry = lo hi color (for discrete style)
        lo/hi = number or min or max = lower/upper bound of subset of range
        color = name of color used for that subset of values
      entry = color (for sequential style)
        color = name of color used for a bin of values
    append arg = yes or no
    bcolor args = type color
      type = bond type or range of types (see below)
      color = name of color or color1/color2/...
    bdiam args = type diam
      type = bond type or range of types (see below)
      diam = diameter of bonds of that type (distance units)
    backcolor arg = color
      color = name of color for background
    boxcolor arg = color
      color = name of color for box lines
    color args = name R G B
      name = name of color
      R,G,B = red/green/blue numeric values from 0.0 to 1.0
    element args = E1 E2 ... EN, where N = # of atom types
      E1,...,EN = element name, e.g. C or Fe or Ga
    every arg = N
      N = dump every this many timesteps
      N can be a variable (see below)
    first arg = yes or no
    format arg = C-style format string for one line of output
    flush arg = yes or no
```

```
image arg = yes or no
label arg = string
  string = character string (e.g. BONDS) to use in header of dump local file
pad arg = Nchar = # of characters to convert timestep to
precision arg = power-of-10 value from 10 to 1000000
region arg = region-ID or "none"
scale arg = yes or no
sort arg = off or id or N or -N
   off = no sorting of per-atom lines within a snapshot
   id = sort per-atom lines by atom ID
   N = sort per-atom lines in ascending order by the Nth column
   -N = sort per-atom lines in descending order by the Nth column
thresh args = attribute operation value
  attribute = same attributes (x,fy,etotal,sxx,etc) used by dump custom style
  operation = "" or ">=" or "==" or "!="
  value = numeric value to compare to
  these 3 args can be replaced by the word "none" to turn off thresholding
unwrap arg = yes or no
```

Examples:

```
dump_modify 1 format "%d %d %20.15g %g %g" scale yes
dump_modify myDump image yes scale no flush yes
dump_modify 1 region mySphere thresh x <0.0 thresh epair >= 3.2
dump_modify xtcdump precision 10000
dump_modify 1 every 1000
dump_modify 1 every v_myVar

dump_modify 1 amap min max cf 0.0 3 min green 0.5 yellow max blue boxcolor red
```

Description:

Modify the parameters of a previously defined dump command. Not all parameters are relevant to all dump styles.

The *acolor* keyword applies only to the dump *image* style. It can be used with the dump image command, when its atom color setting is *type*, to set the color that atoms of each type will be drawn in the image.

The specified *type* should be an integer from 1 to Ntypes = the number of atom types. A wildcard asterisk can be used in place of or in conjunction with the *type* argument to specify a range of atom types. This takes the form "*" or "*n" or "n*" or "m*n". If N = the number of atom types, then an asterisk with no numeric values means all types from 1 to N. A leading asterisk means all types from 1 to n (inclusive). A trailing asterisk means all types from n to N (inclusive). A middle asterisk means all types from m to n (inclusive).

The specified *color* can be a single color which is any of the 140 pre-defined colors (see below) or a color name defined by the dump_modify color option. Or it can be two or more colors separated by a "/" character, e.g. red/green/blue. In the former case, that color is assigned to all the specified atom types. In the latter case, the list of colors are assigned in a round-robin fashion to each of the specified atom types.

The *adiam* keyword applies only to the dump *image* style. It can be used with the dump image command, when its atom diameter setting is *type*, to set the size that atoms of each type will be drawn in the image. The specified *type* should be an integer from 1 to Ntypes. As with the *acolor* keyword, a wildcard asterisk can be used as part of the *type* argument to specify a range of atomt types. The specified *diam* is the size in whatever distance units the input script is using, e.g. Angstroms.

The *amap* keyword applies only to the dump *image* style. It can be used with the dump image command, with its *atom* keyword, when its atom setting is an atom-attribute, to setup a color map. The color map is used to assign a

specific RGB (red/green/blue) color value to an individual atom when it is drawn, based on the atom's attribute, which is a numeric value, e.g. its x−component of velocity if the atom−attribute "vx" was specified.

The basic idea of a color map is that the atom−attribute will be within a range of values, and that range is associated with a a series of colors (e.g. red, blue, green). An atom's specific value (vx = −3.2) can then mapped to the series of colors (e.g. halfway between red and blue), and a specific color is determined via an interpolation procedure.

There are many possible options for the color map, enabled by the *amap* keyword. Here are the details.

The *lo* and *hi* settings determine the range of values allowed for the atom attribute. If numeric values are used for *lo* and/or *hi*, then values that are lower/higher than that value are set to the value. I.e. the range is static. If *lo* is specified as *min* or *hi* as *max* then the range is dynamic, and the lower and/or upper bound will be calculated each time an image is drawn, based on the set of atoms being visualized.

The *style* setting is two letters, such as "ca". The first letter is either "c" for continuous, "d" for discrete, or "s" for sequential. The second letter is either "a" for absolute, or "f" for fractional.

A continuous color map is one in which the color changes continuously from value to value within the range. A discrete color map is one in which discrete colors are assigned to sub−ranges of values within the range. A sequential color map is one in which discrete colors are assigned to a sequence of sub−ranges of values covering the entire range.

An absolute color map is one in which the values to which colors are assigned are specified explicitly as values within the range. A fractional color map is one in which the values to which colors are assigned are specified as a fractional portion of the range. For example if the range is from −10.0 to 10.0, and the color red is to be assigned to atoms with a value of 5.0, then for an absolute color map the number 5.0 would be used. But for a fractional map, the number 0.75 would be used since 5.0 is 3/4 of the way from −10.0 to 10.0.

The *delta* setting is only specified if the style is sequential. It specifies the bin size to use within the range for assigning consecutive colors to. For example, if the range is from −10.0 to 10.0 and a *delta* of 1.0 is used, then 20 colors will be assigned to the range. The first will be from −10.0 <= color1 < −9.0, then 2nd from −9.0 <= color2 < −8.0, etc.

The *N* setting is how many entries follow. The format of the entries depends on whether the color map style is continuous, discrete or sequential. In all cases the *color* setting can be any of the 140 pre−defined colors (see below) or a color name defined by the dump_modify color option.

For continuous color maps, each entry has a *value* and a *color*. The *value* is either a number within the range of values or *min* or *max*. The *value* of the first entry must be *min* and the *value* of the last entry must be *max*. Any entries in between must have increasing values. Note that numeric values can be specified either as absolute numbers or as fractions (0.0 to 1.0) of the range, depending on the "a" or "f" in the style setting for the color map.

Here is how the entries are used to determine the color of an individual atom, given the value X of its atom attribute. X will fall between 2 of the entry values. The color of the atom is linearly interpolated (in each of the RGB values) between the 2 colors associated with those entries. For example, if X = −5.0 and the 2 surrounding entries are "red" at −10.0 and "blue" at 0.0, then the atom's color will be halfway between "red" and "blue", which happens to be "purple".

For discrete color maps, each entry has a *lo* and *hi* value and a *color*. The *lo* and *hi* settings are either numbers within the range of values or *lo* can be *min* or *hi* can be *max*. The *lo* and *hi* settings of the last entry must be *min* and *max*. Other entries can have any *lo* and *hi* values and the sub−ranges of different values can overlap. Note that

numeric *lo* and *hi* values can be specified either as absolute numbers or as fractions (0.0 to 1.0) of the range, depending on the "a" or "f" in the style setting for the color map.

Here is how the entries are used to determine the color of an individual atom, given the value X of its atom attribute. The entries are scanned from first to last. The first time that *lo* <= X <= *hi*, X is assigned the color associated with that entry. You can think of the last entry as assigning a default color (since it will always be matched by X), and the earlier entries as colors that override the default. Also note that no interpolation of a color RGB is done. All atoms will be drawn with one of the colors in the list of entries.

For sequential color maps, each entry has only a *color*. Here is how the entries are used to determine the color of an individual atom, given the value X of its atom attribute. The range is partitioned into N bins of width *binsize*. Thus X will fall in a specific bin from 1 to N, say the Mth bin. If it falls on a boundary between 2 bins, it is considered to be in the higher of the 2 bins. Each bin is assigned a color from the E entries. If E < N, then the colors are repeated. For example if 2 entries with colors red and green are specified, then the odd numbered bins will be red and the even bins green. The color of the atom is the color of its bin. Note that the sequential color map is really a shorthand way of defining a discrete color map without having to specify where all the bin boundaries are.

The *append* keyword applies to all dump styles except *cfg* and *xtc* and *dcd*. It also applies only to text output files, not to binary or gzipped files. If specified as *yes*, then dump snapshots are appended to the end of an existing dump file. If specified as *no*, then a new dump file will be created which will overwrite an existing file with the same name. This keyword can only take effect if the dump_modify command is used after the dump command, but before the first command that causes dump snapshots to be output, e.g. a run or minimize command. Once the dump file has been opened, this keyword has no further effect.

The *bcolor* keyword applies only to the dump *image* style. It can be used with the dump image command, with its *bond* keyword, when its color setting is *type*, to set the color that bonds of each type will be drawn in the image.

The specified *type* should be an integer from 1 to Nbondtypes = the number of bond types. A wildcard asterisk can be used in place of or in conjunction with the *type* argument to specify a range of bond types. This takes the form "*" or "*n" or "n*" or "m*n". If N = the number of bond types, then an asterisk with no numeric values means all types from 1 to N. A leading asterisk means all types from 1 to n (inclusive). A trailing asterisk means all types from n to N (inclusive). A middle asterisk means all types from m to n (inclusive).

The specified *color* can be a single color which is any of the 140 pre−defined colors (see below) or a color name defined by the dump_modify color option. Or it can be two or more colors separated by a "/" character, e.g. red/green/blue. In the former case, that color is assigned to all the specified bond types. In the latter case, the list of colors are assigned in a round−robin fashion to each of the specified bond types.

The *bdiam* keyword applies only to the dump *image* style. It can be used with the dump image command, with its *bond* keyword, when its diam setting is *type*, to set the diameter that bonds of each type will be drawn in the image. The specified *type* should be an integer from 1 to Nbondtypes. As with the *bcolor* keyword, a wildcard asterisk can be used as part of the *type* argument to specify a range of bond types. The specified *diam* is the size in whatever distance units you are using, e.g. Angstroms.

The *backcolor* keyword applies only to the dump *image* style. It sets the background color of the images. The color name can be any of the 140 pre−defined colors (see below) or a color name defined by the dump_modify color option.

The *boxcolor* keyword applies only to the dump *image* style. It sets the color of the simulation box drawn around the atoms in each image. See the "dump image box" command for how to specify that a box be drawn. The color name can be any of the 140 pre−defined colors (see below) or a color name defined by the dump_modify color

option.

The *color* keyword applies only to the dump *image* style. It allows definition of a new color name, in addition to the 140−predefined colors (see below), and associates 3 red/green/blue RGB values with that color name. The color name can then be used with any other dump_modify keyword that takes a color name as a value. The RGB values should each be floating point values between 0.0 and 1.0 inclusive.

When a color name is converted to RGB values, the user−defined color names are searched first, then the 140 pre−defined color names. This means you can also use the *color* keyword to overwrite one of the pre−defined color names with new RBG values.

The *element* keyword applies only to the the dump *cfg* and *image* styles. It associates element names (e.g. H, C, Fe) with LAMMPS atom types. In the case of dump *cfg*, it allows the AtomEye visualization package to read the dump file and render atoms with the appropriate size and color. In the case of dump *image*, the output images will follow the same AtomEye convention. An element name is specified for each atom type (1 to Ntype) in the simulation. The same element name can be given to multiple atom types.

The *every* keyword changes the dump frequency originally specified by the dump command to a new value. The every keyword can be specified in one of two ways. It can be a numeric value in which case it must be > 0. Or it can be an equal−style variable, which should be specified as v_name, where name is the variable name. In this case, the variable is evaluated at the beginning of a run to determine the next timestep at which a dump snapshot will be written out. On that timestep, the variable will be evaluated again to determine the next timestep, etc. Thus the variable should return timestep values. See the stagger() and logfreq() math functions for equal−style variables, as examples of useful functions to use in this context. Other similar math functions could easily be added as options for equal−style variables. When using the variable option with the *every* keyword, you also need to use the *first* option if you want an initial snapshot written to the dump file. The *every* keyword cannot be used with the dump *dcd* style.

For example, the following commands will write snapshots at timesteps 0,10,20,30,100,200,300,1000,2000,etc:

```
variable        s equal logfreq(10,3,10)
dump            1 all atom 100 tmp.dump
dump_modify     1 every v_s first yes
```

The *first* keyword determines whether a dump snapshot is written on the very first timestep after the dump command is invoked. This will always occur if the current timestep is a multiple of N, the frequency specified in the dump command, including timestep 0. But if this is not the case, a dump snapshot will only be written if the setting of this keyword is *yes*. If it is *no*, which is the default, then it will not be written.

The *flush* keyword determines whether a flush operation is invoked after a dump snapshot is written to the dump file. A flush insures the output in that file is current (no buffering by the OS), even if LAMMPS halts before the simulation completes. Flushes cannot be performed with dump style *xtc*.

The text−based dump styles have a default C−style format string which simply specifies %d for integers and %g for real values. The *format* keyword can be used to override the default with a new C−style format string. Do not include a trailing "\n" newline character in the format string. This option has no effect on the *dcd* and *xtc* dump styles since they write binary files. Note that for the *cfg* style, the first two fields (atom id and type) are not actually written into the CFG file, though you must include formats for them in the format string.

The *image* keyword applies only to the dump *atom* style. If the image value is *yes*, 3 flags are appended to each atom's coords which are the absolute box image of the atom in each dimension. For example, an x image flag of −2 with a normalized coord of 0.5 means the atom is in the center of the box, but has passed thru the box

boundary 2 times and is really 2 box lengths to the left of its current coordinate. Note that for dump style *custom* these various values can be printed in the dump file by using the appropriate atom attributes in the dump command itself.

The *label* keyword applies only to the dump *local* style. When it writes local informatoin, such as bond or angle topology to a dump file, it will use the specified *label* to format the header. By default this includes 2 lines:

```
ITEM: NUMBER OF ENTRIES
ITEM: ENTRIES ...
```

The word "ENTRIES" will be replaced with the string specified, e.g. BONDS or ANGLES.

The *pad* keyword only applies when the dump filename is specified with a wildcard "*" character which becomes the timestep. If *pad* is 0, which is the default, the timestep is converted into a string of unpadded length, e.g. 100 or 12000 or 2000000. When *pad* is specified with *Nchar* > 0, the string is padded with leading zeroes so they are all the same length = *Nchar*. For example, pad 7 would yield 0000100, 0012000, 2000000. This can be useful so that post−processing programs can easily read the files in ascending timestep order.

The *precision* keyword only applies to the dump *xtc* style. A specified value of N means that coordinates are stored to 1/N nanometer accuracy, e.g. for N = 1000, the coordinates are written to 1/1000 nanometer accuracy.

The *region* keyword only applies to the dump *custom* and *cfg* and *image* styles. If specified, only atoms in the region will be written to the dump file or included in the image. Only one region can be applied as a filter (the last one specified). See the region command for more details. Note that a region can be defined as the "inside" or "outside" of a geometric shape, and it can be the "union" or "intersection" of a series of simpler regions.

The *scale* keyword applies only to the dump *atom* style. A scale value of *yes* means atom coords are written in normalized units from 0.0 to 1.0 in each box dimension. If the simluation box is triclinic (tilted), then all atom coords will still be between 0.0 and 1.0. A value of *no* means they are written in absolute distance units (e.g. Angstroms or sigma).

The *sort* keyword determines whether lines of per−atom output in a snapshot are sorted or not. A sort value of *off* means they will typically be written in indeterminate order, either in serial or parallel. This is the case even in serial if the atom_modify sort option is turned on, which it is by default, to improve performance. A sort value of *id* means sort the output by atom ID. A sort value of N or −N means sort the output by the value in the Nth column of per−atom info in either ascending or descending order. The dump *local* style cannot be sorted by atom ID, since there are typically multiple lines of output per atom. Some dump styles, such as *dcd* and *xtc*, require sorting by atom ID to format the output file correctly.

IMPORTANT NOTE: Unless it is required by the dump style, sorting dump file output requires extra overhead in terms of CPU and communication cost, as well as memory, versus unsorted output.

The *thresh* keyword only applies to the dump *custom* and *cfg* and *image* styles. Multiple thresholds can be specified. Specifying "none" turns off all threshold criteria. If thresholds are specified, only atoms whose attributes meet all the threshold criteria are written to the dump file or included in the image. The possible attributes that can be tested for are the same as those that can be specified in the dump custom command, with the exception of the *element* attribute, since it is not a numeric value. Note that different attributes can be output by the dump custom command than are used as threshold criteria by the dump_modify command. E.g. you can output the coordinates and stress of atoms whose energy is above some threshold.

The *unwrap* keyword only applies to the dump *dcd* and *xtc* styles. If set to *yes*, coordinates will be written "unwrapped" by the image flags for each atom. Unwrapped means that if the atom has passed thru a periodic

boundary one or more times, the value is printed for what the coordinate would be if it had not been wrapped back into the periodic box. Note that these coordinates may thus be far outside the box size stored with the snapshot.

Restrictions: none

Related commands:

dump, dump image, undump

Default:

The option defaults are

- acolor = * red/green/blue/yellow/aqua/cyan
- adiam = * 1.0
- amap = min max cf 2 min blue max red
- append = no
- bcolor = * red/green/blue/yellow/aqua/cyan
- bdiam = * 0.5
- backcolor = black
- boxcolor = yellow
- color = 140 color names are pre−defined as listed below
- element = "C" for every atom type
- every = whatever it was set to via the dump command
- first = no
- flush = yes
- format = %d and %g for each integer or floating point value
- image = no
- label = ENTRIES
- pad = 0
- precision = 1000
- region = none
- scale = yes
- sort = off for dump styles *atom*, *custom*, *cfg*, and *local*
- sort = id for dump styles *dcd*, *xtc*, and *xyz*
- thresh = none
- unwrap = no

These are the 140 colors that LAMMPS pre−defines for use with the dump image and dump_modify commands. Additional colors can be defined with the dump_modify color command. The 3 numbers listed for each name are the RGB (red/green/blue) values. Divide each value by 255 to get the equivalent 0.0 to 1.0 value.

aliceblue = 240, 248, 255	antiquewhite = 250, 235, 215	aqua = 0, 255, 255	aquamarine = 127, 255, 212	azure = 240, 255, 255
beige = 245, 245, 220	bisque = 255, 228, 196	black = 0, 0, 0	blanchedalmond = 255, 255, 205	blue = 0, 0, 255
blueviolet = 138, 43, 226	brown = 165, 42, 42	burlywood = 222, 184, 135	cadetblue = 95, 158, 160	chartreuse = 127, 255, 0
chocolate = 210, 105, 30	coral = 255, 127, 80	cornflowerblue = 100, 149, 237	cornsilk = 255, 248, 220	crimson = 220, 20, 60
cyan = 0, 255, 255	darkblue = 0, 0, 139	darkcyan = 0, 139, 139		

			darkgoldenrod = 184, 134, 11	darkgray = 169, 169, 169
darkgreen = 0, 100, 0	darkkhaki = 189, 183, 107	darkmagenta = 139, 0, 139	darkolivegreen = 85, 107, 47	darkorange = 255, 140, 0
darkorchid = 153, 50, 204	darkred = 139, 0, 0	darksalmon = 233, 150, 122	darkseagreen = 143, 188, 143	darkslateblue = 72, 61, 139
darkslategray = 47, 79, 79	darkturquoise = 0, 206, 209	darkviolet = 148, 0, 211	deeppink = 255, 20, 147	deepskyblue = 0, 191, 255
dimgray = 105, 105, 105	dodgerblue = 30, 144, 255	firebrick = 178, 34, 34	floralwhite = 255, 250, 240	forestgreen = 34, 139, 34
fuchsia = 255, 0, 255	gainsboro = 220, 220, 220	ghostwhite = 248, 248, 255	gold = 255, 215, 0	goldenrod = 218, 165, 32
gray = 128, 128, 128	green = 0, 128, 0	greenyellow = 173, 255, 47	honeydew = 240, 255, 240	hotpink = 255, 105, 180
indianred = 205, 92, 92	indigo = 75, 0, 130	ivory = 255, 240, 240	khaki = 240, 230, 140	lavender = 230, 230, 250
lavenderblush = 255, 240, 245	lawngreen = 124, 252, 0	lemonchiffon = 255, 250, 205	lightblue = 173, 216, 230	lightcoral = 240, 128, 128
lightcyan = 224, 255, 255	lightgoldenrodyellow = 250, 250, 210	lightgreen = 144, 238, 144	lightgrey = 211, 211, 211	lightpink = 255, 182, 193
lightsalmon = 255, 160, 122	lightseagreen = 32, 178, 170	lightskyblue = 135, 206, 250	lightslategray = 119, 136, 153	lightsteelblue = 176, 196, 222
lightyellow = 255, 255, 224	lime = 0, 255, 0	limegreen = 50, 205, 50	linen = 250, 240, 230	magenta = 255, 0, 255
maroon = 128, 0, 0	mediumaquamarine = 102, 205, 170	mediumblue = 0, 0, 205	mediumorchid = 186, 85, 211	mediumpurple = 147, 112, 219
mediumseagreen = 60, 179, 113	mediumslateblue = 123, 104, 238	mediumspringgreen = 0, 250, 154	mediumturquoise = 72, 209, 204	mediumvioletred = 199, 21, 133
midnightblue = 25, 25, 112	mintcream = 245, 255, 250	mistyrose = 255, 228, 225	moccasin = 255, 228, 181	navajowhite = 255, 222, 173
navy = 0, 0, 128	oldlace = 253, 245, 230	olive = 128, 128, 0	olivedrab = 107, 142, 35	orange = 255, 165, 0
orangered = 255, 69, 0	orchid = 218, 112, 214	palegoldenrod = 238, 232, 170	palegreen = 152, 251, 152	paleturquoise = 175, 238, 238
palevioletred = 219, 112, 147	papayawhip = 255, 239, 213	peachpuff = 255, 239, 213	peru = 205, 133, 63	pink = 255, 192, 203
plum = 221, 160, 221	powderblue = 176, 224, 230	purple = 128, 0, 128	red = 255, 0, 0	rosybrown = 188, 143, 143
royalblue = 65, 105, 225	saddlebrown = 139, 69, 19	salmon = 250, 128, 114	sandybrown = 244, 164, 96	seagreen = 46, 139, 87
seashell = 255, 245, 238	sienna = 160, 82, 45	silver = 192, 192, 192	skyblue = 135, 206, 235	slateblue = 106, 90, 205
slategray = 112, 128, 144	snow = 255, 250, 250	springgreen = 0, 255, 127	steelblue = 70, 130, 180	tan = 210, 180, 140
teal = 0, 128, 128	thistle = 216, 191, 216	tomato = 253, 99, 71	turquoise = 64, 224, 208	violet = 238, 130, 238
	white = 255, 255, 255		yellow = 255, 255, 0	

wheat = 245, 222, 179		whitesmoke = 245, 245, 245		yellowgreen = 154, 205, 50

echo command

Syntax:

```
echo style
```

- style = *none* or *screen* or *log* or *both*

Examples:

```
echo both
echo log
```

Description:

This command determines whether LAMMPS echoes each input script command to the screen and/or log file as it is read and processed. If an input script has errors, it can be useful to look at echoed output to see the last command processed.

The command−line switch −echo can be used in place of this command.

Restrictions: none

Related commands: none

Default:

```
echo log
```

fix command

Syntax:

```
fix ID group-ID style args
```

- ID = user−assigned name for the fix
- group−ID = ID of the group of atoms to apply the fix to
- style = one of a long list of possible style names (see below)
- args = arguments used by a particular style

Examples:

```
fix 1 all nve
fix 3 all nvt temp 300.0 300.0 0.01
fix mine top setforce 0.0 NULL 0.0
```

Description:

Set a fix that will be applied to a group of atoms. In LAMMPS, a "fix" is any operation that is applied to the system during timestepping or minimization. Examples include updating of atom positions and velocities due to time integration, controlling temperature, applying constraint forces to atoms, enforcing boundary conditions, computing diagnostics, etc. There are dozens of fixes defined in LAMMPS and new ones can be added; see this section for a discussion.

Fixes perform their operations at different stages of the timestep. If 2 or more fixes operate at the same stage of the timestep, they are invoked in the order they were specified in the input script.

The ID of a fix can only contain alphanumeric characters and underscores.

Fixes can be deleted with the unfix command.

IMPORTANT NOTE: The unfix command is the only way to turn off a fix; simply specifying a new fix with a similar style will not turn off the first one. This is especially important to realize for integration fixes. For example, using a fix nve command for a second run after using a fix nvt command for the first run, will not cancel out the NVT time integration invoked by the "fix nvt" command. Thus two time integrators would be in place!

If you specify a new fix with the same ID and style as an existing fix, the old fix is deleted and the new one is created (presumably with new settings). This is the same as if an "unfix" command were first performed on the old fix, except that the new fix is kept in the same order relative to the existing fixes as the old one originally was. Note that this operation also wipes out any additional changes made to the old fix via the fix_modify command.

The fix modify command allows settings for some fixes to be reset. See the doc page for individual fixes for details.

Some fixes store an internal "state" which is written to binary restart files via the restart or write_restart commands. This allows the fix to continue on with its calculations in a restarted simulation. See the read_restart command for info on how to re−specify a fix in an input script that reads a restart file. See the doc pages for individual fixes for info on which ones can be restarted.

Some fixes calculate one of three styles of quantities: global, per−atom, or local, which can be used by other commands or output as described below. A global quantity is one or more system−wide values, e.g. the energy of a wall interacting with particles. A per−atom quantity is one or more values per atom, e.g. the displacement vector for each atom since time 0. Per−atom values are set to 0.0 for atoms not in the specified fix group. Local quantities are calculated by each processor based on the atoms it owns, but there may be zero or more per atoms.

Note that a single fix may produces either global or per−atom or local quantities (or none at all), but never more than one of these.

Global, per−atom, and local quantities each come in three kinds: a single scalar value, a vector of values, or a 2d array of values. The doc page for each fix describes the style and kind of values it produces, e.g. a per−atom vector. Some fixes produce more than one kind of a single style, e.g. a global scalar and a global vector.

When a fix quantity is accessed, as in many of the output commands discussed below, it can be referenced via the following bracket notation, where ID is the ID of the fix:

f_ID	entire scalar, vector, or array
f_ID[I]	one element of vector, one column of array
f_ID[I][J]	one element of array

In other words, using one bracket reduces the dimension of the quantity once (vector −> scalar, array −> vector). Using two brackets reduces the dimension twice (array −> scalar). Thus a command that uses scalar fix values as input can also process elements of a vector or array.

Note that commands and variables which use fix quantities typically do not allow for all kinds, e.g. a command may require a vector of values, not a scalar. This means there is no ambiguity about referring to a fix quantity as f_ID even if it produces, for example, both a scalar and vector. The doc pages for various commands explain the details.

In LAMMPS, the values generated by a fix can be used in several ways:

- Global values can be output via the thermo_style custom or fix ave/time command. Or the values can be referenced in a variable equal or variable atom command.
- Per−atom values can be output via the dump custom command or the fix ave/spatial command. Or they can be time−averaged via the fix ave/atom command or reduced by the compute reduce command. Or the per−atom values can be referenced in an atom−style variable.
- Local values can be reduced by the compute reduce command, or histogrammed by the fix ave/histo command.

See this howto section for a summary of various LAMMPS output options, many of which involve fixes.

The results of fixes that calculate global quantities can be either "intensive" or "extensive" values. Intensive means the value is independent of the number of atoms in the simulation, e.g. temperature. Extensive means the value scales with the number of atoms in the simulation, e.g. total rotational kinetic energy. Thermodynamic output will normalize extensive values by the number of atoms in the system, depending on the "thermo_modify norm" setting. It will not normalize intensive values. If a fix value is accessed in another way, e.g. by a variable, you may want to know whether it is an intensive or extensive value. See the doc page for individual fixes for further info.

Each fix style has its own documentation page which describes its arguments and what it does, as listed below. Here is an alphabetic list of fix styles available in LAMMPS:

- adapt − change a simulation parameter over time
- addforce − add a force to each atom
- aveforce − add an averaged force to each atom
- ave/atom − compute per−atom time−averaged quantities
- ave/histo − compute/output time−averaged histograms
- ave/spatial − compute/output time−averaged per−atom quantities by layer
- ave/time − compute/output global time−averaged quantities
- bond/break − break bonds on the fly
- bond/create − create bonds on the fly
- bond/swap − Monte Carlo bond swapping
- box/relax − relax box size during energy minimization
- deform − change the simulation box size/shape
- deposit − add new atoms above a surface
- drag − drag atoms towards a defined coordinate
- dt/reset − reset the timestep based on velocity, forces
- efield − impose electric field on system
- enforce2d − zero out z−dimension velocity and force
- evaporate − remove atoms from simulation periodically
- external − callback to an external driver program
- freeze − freeze atoms in a granular simulation
- gravity − add gravity to atoms in a granular simulation
- gcmc − grand canonical insertions/deletions
- heat − add/subtract momentum−conserving heat
- indent − impose force due to an indenter
- langevin − Langevin temperature control
- lineforce − constrain atoms to move in a line
- momentum − zero the linear and/or angular momentum of a group of atoms
- move − move atoms in a prescribed fashion
- msst − multi−scale shock technique (MSST) integration
- neb − nudged elastic band (NEB) spring forces
- nph − constant NPH time integration via Nose/Hoover
- nph/asphere − NPH for aspherical particles
- nph/sphere − NPH for spherical particles
- npt − constant NPT time integration via Nose/Hoover
- npt/asphere − NPT for aspherical particles
- npt/sphere − NPT for spherical particles
- nve − constant NVE time integration
- nve/asphere − NVT for aspherical particles
- nve/limit − NVE with limited step length
- nve/noforce − NVE without forces (v only)
- nve/sphere − NVT for spherical particles
- nvt − constant NVT time integration via Nose/Hoover
- nvt/asphere − NVT for aspherical particles
- nvt/sllod − NVT for NEMD with SLLOD equations
- nvt/sphere − NVT for spherical particles
- orient/fcc − add grain boundary migration force
- planeforce − constrain atoms to move in a plane
- poems − constrain clusters of atoms to move as coupled rigid bodies
- pour − pour new atoms into a granular simulation domain
- press/berendsen − pressure control by Berendsen barostat
- print − print text and variables during a simulation
- reax/bonds − write out ReaxFF bond information recenter − constrain the center−of−mass position of a

group of atoms
- restrain − constrain a bond, angle, dihedral
- rigid − constrain one or more clusters of atoms to move as a rigid body with NVE integration
- rigid/nve − constrain one or more clusters of atoms to move as a rigid body with alternate NVE integration
- rigid/nvt − constrain one or more clusters of atoms to move as a rigid body with NVT integration
- setforce − set the force on each atom
- shake − SHAKE constraints on bonds and/or angles
- spring − apply harmonic spring force to group of atoms
- spring/rg − spring on radius of gyration of group of atoms
- spring/self − spring from each atom to its origin
- srd − stochastic rotation dynamics (SRD)
- store/force − store force on each atom
- store/state − store attributes for each atom
- temp/berendsen − temperature control by Berendsen thermostat
- temp/rescale − temperature control by velocity rescaling
- thermal/conductivity − Muller−Plathe kinetic energy exchange for thermal conductivity calculation
- tmd − guide a group of atoms to a new configuration
- ttm − two−temperature model for electronic/atomic coupling
- viscosity − Muller−Plathe momentum exchange for viscosity calculation
- viscous − viscous damping for granular simulations
- wall/colloid − Lennard−Jones wall interacting with finite−size particles
- wall/gran − frictional wall(s) for granular simulations
- wall/harmonic − harmonic spring wall
- wall/lj126 − Lennard−Jones 12−6 wall
- wall/lj93 − Lennard−Jones 9−3 wall
- wall/reflect − reflecting wall(s)
- wall/region − use region surface as wall
- wall/srd − slip/no−slip wall for SRD particles

There are also additional fix styles submitted by users which are included in the LAMMPS distribution. The list of these with links to the individual styles are given in the fix section of this page.

There are also additional accelerated fix styles included in the LAMMPS distribution for faster performance on CPUs and GPUs. The list of these with links to the individual styles are given in the pair section of this page.

Restrictions:

Some fix styles are part of specific packages. They are only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info on packages. The doc pages for individual fixes tell if it is part of a package.

Related commands:

unfix, fix_modify

Default: none

fix adapt command

Syntax:

```
fix ID group-ID adapt N attribute args ... keyword value ...
```

- ID, group−ID are documented in fix command
- adapt = style name of this fix command
- N = adapt simulation settings every this many timesteps
- one or more attribute/arg pairs may be appended
- attribute = *pair* or *kspace* or *atom*

```
pair args = pstyle pparam I J v_name
    pstyle = pair style name, e.g. lj/cut
    pparam = parameter to adapt over time
    I,J = type pair(s) to set parameter for
    v_name = variable with name that calculates value of pparam
  kspace arg = v_name
    v_name = variable with name that calculates scale factor on K-space terms
  atom args = aparam v_name
    aparam = parameter to adapt over time
    v_name = variable with name that calculates value of aparam
```

- zero or more keyword/value pairs may be appended
- keyword = *scale* or *reset*

```
scale value = no or yes
    no = the variable value is the new setting
    yes = the variable value multiplies the original setting

reset value = no or yes
    no = values will remain altered at the end of a run
    yes = reset altered values to their original values at the end of a run
```

Examples:

```
fix 1 all adapt 1 pair soft a 1 1 v_prefactor
fix 1 all adapt 1 pair soft a 2* 3 v_prefactor
fix 1 all adapt 1 pair lj/cut epsilon * * v_scale1 coul/cut scale 3 3 v_scale2 scale yes reset yes
fix 1 all adapt 10 atom diameter v_size
```

Description:

Change or adapt one or more specific simulation attributes or settings over time as a simulation runs. Pair potential and K−space and atom attributes which can be varied by this fix are discussed below. Many other fixes can also be used to time−vary simulation parameters, e.g. the "fix deform" command will change the simulation box size/shape and the "fix move" command will change atom positions and velocities in a prescribed manner.

If *N* is specified as 0, the specified attributes are only changed once, before the simulation begins. This is all that is needed if the associated variables are not time−dependent. If $N > 0$, then changes are made every *N* steps during the simulation, presumably with a variable that is time−dependent.

Depending on the value of the *reset* keyword, attributes changed by this fix will or will not be reset back to their original values at the end of a simulation. Even if *reset* is specified as *yes*, a restart file written during a simulation will contain the modified settings.

IMPORTANT NOTE: Currently, only the *pair* and *kspace* params are resettable. *Atom* attributes are not. This will be added at some point.

If the *scale* keyword is set to *no*, then the value the parameter is set to will be whatever the variable generates. If the *scale* keyword is set to *yes*, then the value of the altered parameter will be the initial value of that parameter multiplied by whatever the variable generates. I.e. the variable is now a "scale factor" applied in (presumably) a time−varying fashion to the parameter. Internally, the parameters themselves are actually altered; make sure you use the *reset yes* option if you want the parameters to be restored to their initial values after the run.

The *pair* keyword enables various parameters of potentials defined by the pair_style command to be changed, if the pair style supports it. Note that the pair_style and pair_coeff commands must be used in the usual manner to specify these parameters initially; the fix adapt command simply overrides the parameters.

The *pstyle* argument is the name of the pair style. If pair_style hybrid or hybrid/overlay is used, *pstyle* should be a sub−style name. For example, *pstyle* could be specified as "soft" or "lubricate". The *pparam* argument is the name of the parameter to change. This is the current list of pair styles and parameters that can be varied by this fix. See the doc pages for individual pair styles and their energy formulas for the meaning of these parameters:

born	a,b,c	type pairs
buck	a,c	type pairs
coul/cut	scale	type pairs
coul/debye	scale	type pairs
coul/long	scale	type pairs
lj/cut	epsilon	type pairs
lj/cut/opt	epsilon	type pairs
lubricate	mu	global
gauss	a	type pairs
soft	a	type pairs

IMPORTANT NOTE: It is easy to add new potentials and their parameters to this list. All it typically takes is adding an extract() method to the pair_*.cpp file associated with the potential.

Some parameters are global settings for the pair style, e.g. the viscosity setting "mu" for pair_style lubricate. Other parameters apply to atom type pairs within the pair style, e.g. the prefactor "a" for pair_style soft.

Note that for many of the potentials, the parameter that can be varied is effectively a prefactor on the entire energy expression for the potential, e.g. the lj/cut epsilon. The parameters listed as "scale" are exactly that, since the energy expression for the coul/cut potential (for example) has no labeled prefactor in its formula. To apply an effective prefactor to some potentials, multiple parameters need to be altered. For example, the Buckingham potential needs both the A and C terms altered together. To scale the Buckingham potential, you should thus list the pair style twice, once for A and once for C.

If a type pair parameter is specified, the *I* and *J* settings should be specified to indicate which type pairs to apply it to. If a global parameter is specified, the *I* and *J* settings still need to be specified, but are ignored.

Similar to the pair_coeff command, I and J can be specified in one of two ways. Explicit numeric values can be used for each, as in the 1st example above. I <= J is required. LAMMPS sets the coefficients for the symmetric J,I interaction to the same values.

A wild−card asterisk can be used in place of or in conjunction with the I,J arguments to set the coefficients for multiple pairs of atom types. This takes the form "*" or "*n" or "n*" or "m*n". If N = the number of atom types,

then an asterisk with no numeric values means all types from 1 to N. A leading asterisk means all types from 1 to n (inclusive). A trailing asterisk means all types from n to N (inclusive). A middle asterisk means all types from m to n (inclusive). Note that only type pairs with I <= J are considered; if asterisks imply type pairs where J < I, they are ignored.

IMPROTANT NOTE: If pair_style hybrid or hybrid/overlay is being used, then the *pstyle* will be a sub−style name. You must specify I,J arguments that correspond to type pair values defined (via the pair_coeff command) for that sub−style.

The *v_name* argument for keyword *pair* is the name of an equal−style variable which will be evaluated each time this fix is invoked to set the parameter to a new value. It should be specified as v_name, where name is the variable name. Equal−style variables can specify formulas with various mathematical functions, and include thermo_style command keywords for the simulation box parameters and timestep and elapsed time. Thus it is easy to specify parameters that change as a function of time or span consecutive runs in a continuous fashion. For the latter, see the *start* and *stop* keywords of the run command and the *elaplong* keyword of thermo_style custom for details.

For example, these commands would change the prefactor coefficient of the pair_style soft potential from 10.0 to 30.0 in a linear fashion over the course of a simulation:

```
variable prefactor equal ramp(10,30)
fix 1 all adapt 1 pair soft a * * v_prefactor
```

The *kspace* keyword used the specified variable as a scale factor on the energy, forces, virial calculated by whatever K−Space solver is defined by the kspace_style command. If the variable has a value of 1.0, then the solver is unaltered.

The *kspace* keyword works this way whether the *scale* keyword is set to *no* or *yes*.

The *atom* keyword enables various atom properties to be changed. The *aparam* argument is the name of the parameter to change. This is the current list of atom parameters that can be varied by this fix:

- diameter = diameter of particle

The *v_name* argument of the *atom* keyword is the name of an equal−style variable which will be evaluated each time this fix is invoked to set the parameter to a new value. It should be specified as v_name, where name is the variable name. See the discussion above describing the formulas associated with equal−style variables. The new value is assigned to the corresponding attribute for all atoms in the fix group.

If the atom parameter is *diameter* and per−atom density and per−atom mass are defined for particles (e.g. atom_style granular), then the mass of each particle is also changed when the diameter changes (density is assumed to stay constant).

For example, these commands would shrink the diameter of all granular particles in the "center" group from 1.0 to 0.1 in a linear fashion over the course of a 1000−step simulation:

```
variable size equal ramp(1.0,0.1)
fix 1 center adapt 10 atom diameter v_size
```

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix. No global or per−atom quantities are stored by this fix for access by various output commands. No parameter

of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions: none

Related commands:

compute ti

Default:

The option defaults are scale = no, reset = no.

fix addforce command

fix addforce/cuda command

Syntax:

```
fix ID group-ID addforce fx fy fz keyword value ...
```

- ID, group−ID are documented in fix command
- addforce = style name of this fix command
- fx,fy,fz = force component values (force units)
- any of fx,fy,fz can be a variable (see below)
- zero or more keyword/value pairs may be appended to args
- keyword = *region* or *energy*

```
  region value = region-ID
      region-ID = ID of region atoms must be in to have added force
    energy value = v_name
      v_name = variable with name that calculates the potential energy of each atom in the added
```

Examples:

```
fix kick flow addforce 1.0 0.0 0.0
fix kick flow addforce 1.0 0.0 v_oscillate
fix ff boundary addforce 0.0 0.0 v_push energy v_espace
```

Description:

Add fx,fy,fz to the corresponding component of force for each atom in the group. This command can be used to give an additional push to atoms in a simulation, such as for a simulation of Poiseuille flow in a channel.

Any of the 3 quantities defining the force components can be specified as an equal−style or atom−style variable, namely *fx*, *fy*, *fz*. If the value is a variable, it should be specified as v_name, where name is the variable name. In this case, the variable will be evaluated each timestep, and its value used to determine the force component.

Equal−style variables can specify formulas with various mathematical functions, and include thermo_style command keywords for the simulation box parameters and timestep and elapsed time. Thus it is easy to specify a time−dependent force field.

Atom−style variables can specify the same formulas as equal−style variables but can also include per−atom values, such as atom coordinates. Thus it is easy to specify a spatially−dependent force field with optional time−dependence as well.

If the *region* keyword is used, the atom must also be in the specified geometric region in order to have force added to it.

Adding a force to atoms implies a change in their potential energy as they move due to the applied force field. For dynamics via the "run" command, this energy can be optionally added to the system's potential energy for thermodynamic output (see below). For energy minimization via the "minimize" command, this energy must be added to the system's potential energy to formulate a self−consistent minimization problem (see below).

The *energy* keyword is not allowed if the added force is a constant vector F = (fx,fy,fz), with all components defined as numeric constants and not as variables. This is because LAMMPS can compute the energy for each atom directly as E = −x dot F = −(x*fx + y*fy + z*fz), so that −Grad(E) = F.

The *energy* keyword is optional if the added force is defined with one or more variables, and if you are performing dynamics via the run command. If the keyword is not used, LAMMPS will set the energy to 0.0, which is typically fine for dynamics.

The *energy* keyword is required if the added force is defined with one or more variables, and you are performing energy minimization via the "minimize" command. The keyword specifies the name of an atom−style variable which is used to compute the energy of each atom as function of its position. Like variables used for *fx*, *fy*, *fz*, the energy variable is specified as v_name, where name is the variable name.

Note that when the *energy* keyword is used during an energy minimization, you must insure that the formula defined for the atom−style variable is consistent with the force variable formulas, i.e. that −Grad(E) = F. For example, if the force were a spring−like F = kx, then the energy formula should be E = −0.5kx^2. If you don't do this correctly, the minimization will not converge properly.

Styles with a *cuda* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in this section of the manual. The accelerated styles take the same arguments and should produce the same results, except for round−off and precision issues.

These accelerated styles are part of the "user−cuda" package. They are only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the −suffix command−line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See this section of the manual for more instructions on how to use the accelerated styles effectively.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files.

The fix_modify *energy* option is supported by this fix to add the potential "energy" inferred by the added force to the system's potential energy as part of thermodynamic output. This is a fictitious quantity but is needed so that the minimize command can include the forces added by this fix in a consistent manner. I.e. there is a decrease in potential energy when atoms move in the direction of the added force.

This fix computes a global scalar and a global 3−vector of forces, which can be accessed by various output commands. The scalar is the potential energy discussed above. The vector is the total force on the group of atoms before the forces on individual atoms are changed by the fix. The scalar and vector values calculated by this fix are "extensive".

No parameter of this fix can be used with the *start/stop* keywords of the run command.

The forces due to this fix are imposed during an energy minimization, invoked by the minimize command. You should not specify force components with a variable that has time−dependence for use with a minimizer, since the minimizer increments the timestep as the iteration count during the minimization.

IMPORTANT NOTE: If you want the fictitious potential energy associated with the added forces to be included in the total potential energy of the system (the quantity being minimized), you MUST enable the fix_modify *energy* option for this fix.

Restrictions: none

Related commands:

fix setforce, fix aveforce

Default: none

fix addtorque command

Syntax:

```
fix ID group-ID addtorque Tx Ty Tz
```

- ID, group−ID are documented in fix command
- addtorque = style name of this fix command
- Tx,Ty,Tz = torque component values (torque units)
- any of Tx,Ty,Tz can be a variable (see below)

Examples:

```
fix kick bead addtorque 2.0 3.0 5.0
fix kick bead addtorque 0.0 0.0 v_oscillate
```

Description:

Add a set of forces to each atom in the group such that:

- the components of the total torque applied on the group (around its center of mass) are Tx,Ty,Tz
- the group would move as a rigid body in the absence of other forces.

This command can be used to drive a group of atoms into rotation.

Any of the 3 quantities defining the torque components can be specified as an equal−style variable, namely *Tx*, *Ty*, *Tz*. If the value is a variable, it should be specified as v_name, where name is the variable name. In this case, the variable will be evaluated each timestep, and its value used to determine the torque component.

Equal−style variables can specify formulas with various mathematical functions, and include thermo_style command keywords for the simulation box parameters and timestep and elapsed time. Thus it is easy to specify a time−dependent torque.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files.

The fix_modify *energy* option is supported by this fix to add the potential "energy" inferred by the added forces to the system's potential energy as part of thermodynamic output. This is a fictitious quantity but is needed so that the minimize command can include the forces added by this fix in a consistent manner. I.e. there is a decrease in potential energy when atoms move in the direction of the added forces.

This fix computes a global scalar and a global 3−vector, which can be accessed by various output commands. The scalar is the potential energy discussed above. The vector is the total torque on the group of atoms before the forces on individual atoms are changed by the fix. The scalar and vector values calculated by this fix are "extensive".

No parameter of this fix can be used with the *start/stop* keywords of the run command.

The forces due to this fix are imposed during an energy minimization, invoked by the minimize command. You

should not specify force components with a variable that has time−dependence for use with a minimizer, since the minimizer increments the timestep as the iteration count during the minimization.

Restrictions:

This fix is part of the "user−misc" package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

Related commands:

fix addforce

Default: none

fix atc command

Syntax:

```
fix ID groupID atc type paramfile
```

- ID, group−ID are documented in fix command
- atc = style name of this fix command
- type = *thermal* or *two_temperature* or *hardy*

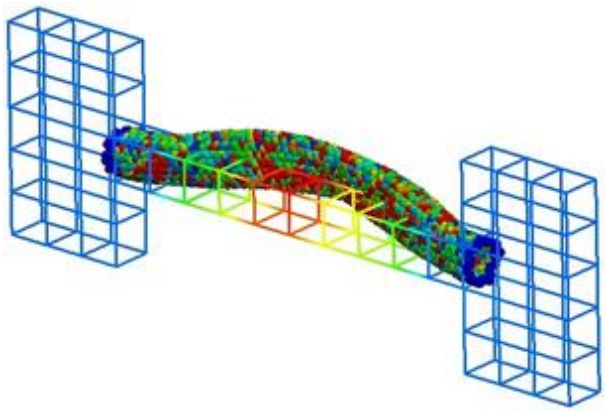
  ```
  thermal = thermal coupling with field: temperature
   two_temperature = electron−phonon coupling with field, temperature and electron_temperature
   hardy = Hardy on−the−fly post−processing
  ```
- paramfile = file with material parameters (not specified for *hardy* type)

Examples:

```
fix AtC atc_atoms atc thermal Ar_thermal.dat
fix AtC atc_atoms atc transfer hardy
```

Description:

This fix creates a coupled finite element (FE) and molecular dynamics (MD) simulation and/or an on−the−fly estimation of continuum fields, where a FE mesh is specified and overlaps the particles, something like this:



Interscale operators are defined that construct continuum fields from atomic data. Coupled simulations use FE projection approximated on a discrete field. Currently, coupling is restricted to thermal physics. The Hardy module can use either FE projection or integration Kernels evaluated at mesh points.

Coupling methods enable appropriate corrections to the atomic data to be made based on the FE field. For example, a Gaussian isokinetic thermostat can apply heat sources to the atoms that varies in space on the same scale as the FE element size. Meshes are not created automatically and must be specified on LAMMPS regions with prescribed element sizes.

Coupling and post−processing can be combined in the same simulations using separate fix atc commands.

Note that mesh computations and storage run in serial (not parallelized) so performance will degrade when large element counts are used.

For detailed exposition of the theory and algorithms implemented in this fix, please see the papers here and here. Please refer to the standard finite element (FE) texts, such as this book, for the basics of FE simulation.

Thermal and *two_temperature* (coupling) types use a Verlet time−integration algorithm. The *hardy* type does not contain its own time−integrator and must be used with a separate fix that does contain one, e.g. fix nve, fix nvt, etc.

A set of example input files with the attendant material files are included in the examples/USER/atc directory of the LAMMPS distribution.

An extensive set of additional documentation pages for the options turned on via the fix_modify command for this fix are inlcluded in the doc/USER/atc directory of the LAMMPS distribution. Individual doc pages are listed and linked to below.

The following commands are typical of a coupling problem:

```
# ... commands to create and initialize the MD system

# initial fix to designate coupling type and group to apply it to
# tag group physics material_file
fix AtC internal atc thermal Ar_thermal.mat

# create a uniform 12 x 2 x 2 mesh that covers region contain the group
# nx ny nz region periodicity
fix_modify AtC fem create mesh 12 2 2 mdRegion f p p

# specify the control method for the type of coupling
# physics control_type
fix_modify AtC transfer thermal control flux

# specify the initial values for the empirical field "temperature"
# field node_group value
fix_modify AtC transfer initial temperature all 30.0

# create an output stream for nodal fields
# filename output_frequency
fix_modify AtC transfer output atc_fe_output 100

run 1000
```

The following commands are typical of a post−processing (Hardy) problem:

```
# ... commands to create and initialize the MD system

# initial fix to designate post−processing and the group to apply it to
# no material file is allowed nor required
fix AtC internal atc hardy

# create a uniform 1 x 1 x 1 mesh that covers region contain the group
# with periodicity this effectively creats a system average
fix_modify AtC fem create mesh 1 1 1 box p p p

# change from default lagrangian map to eulerian
# refreshed every 100 steps
fix_modify AtC atom_element_map eulerian 100

# start with no field defined
fix_modify AtC transfer fields none
```

```
# add mass density, potential energy density, stress and temperature
fix_modify AtC transfer fields add density energy stress temperature

# create an output stream for nodal fields
# filename output_frequency
fix_modify AtC transfer output nvtFE 100 text

run 1000
```

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. The fix_modify options relevant to this fix are listed below. No global scalar or vector or per−atom quantities are stored by this fix for access by various output commands. No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions:

This fix is part of the "user−atc" package. It is only enabled if LAMMPS was built with that package, which also requires the ATC library be built and linked with LAMMPS. See the Making LAMMPS section for more info.

Related commands:

After specifying this fix in your input script, several other fix_modify commands are used to setup the problem, e.g. define the finite element mesh and prescribe initial and boundary conditions.

fix_modify commands for setup:

- fix_modify AtC fem create mesh
- fix_modify AtC mesh create_nodeset
- fix_modify AtC mesh create_faceset
- fix_modify AtC mesh create_elementset
- fix_modify AtC transfer internal
- fix_modify AtC transfer boundary
- fix_modify AtC transfer internal_quadrature
- fix_modify AtC transfer pmfc
- fix_modify AtC extrinsic electron_integration

fix_modify commands for boundary and initial conditions:

- fix_modify AtC transfer initial
- fix_modify AtC transfer fix
- fix_modify AtC transfer unfix
- fix_modify AtC transfer fix_flux
- fix_modify AtC transferunfix_flux
- fix_modify AtC transfer source
- fix_modify AtC transfer remove_source

fix_modify commands for control and filtering:

- fix_modify AtC transfer thermal control
- fix_modify AtC transfer filter
- fix_modify AtC transfer filter scale

- fix_modify AtC transfer equilibrium_start
- fix_modify AtC extrinsic exchange

fix_modify commands for output:

- fix_modify AtC transfer output
- fix_modify AtC transfer atomic_output
- fix_modify AtC mesh output
- fix_modify AtC transfer write_restart
- fix_modify AtC transfer read_restart

fix_modify commands for post−processing:

- fix_modify AtC transfer fields
- fix_modify AtC transfer gradients
- fix_modify AtC transfer rates
- fix_modify AtC transfer computes
- fix_modify AtC set
- fix_modify AtC transfer on_the_fly
- fix_modify AtC boundary_integral
- fix_modify AtC contour_integral

miscellaneous fix_modify commands:

- fix_modify AtC transfer atom_element_map
- fix_modify AtC transfer neighbor_reset_frequency

Default: none

(Wagner) Wagner, Jones, Templeton, Parks, Special Issue of Computer Methods and Applied Mechanics, 197, 3351−3365 (2008).

(Zimmerman) Zimmerman, Webb, Hoyt, Jones, Klein, Bammann, Special Issue of Modelling and Simulation in Materials Science and Engineering, 12, S319 (2004).

(Hughes) T.J.R Hughes, "The Finite Element Method," Dover (2003).

fix ave/atom command

Syntax:

```
fix ID group-ID ave/atom Nevery Nrepeat Nfreq value1 value2 ...
```

- ID, group−ID are documented in fix command
- ave/atom = style name of this fix command
- Nevery = use input values every this many timesteps
- Nrepeat = # of times to use input values for calculating averages
- Nfreq = calculate averages every this many timesteps one or more input values can be listed
- value = x, y, z, vx, vy, vz, fx, fy, fz, c_ID, c_ID[i], f_ID, f_ID[i], v_name

```
x,y,z,vx,vy,vz,fx,fy,fz = atom attribute (position, velocity, force component)
c_ID = per-atom vector calculated by a compute with ID
c_ID[I] = Ith column of per-atom array calculated by a compute with ID
f_ID = per-atom vector calculated by a fix with ID
f_ID[I] = Ith column of per-atom array calculated by a fix with ID
v_name = per-atom vector calculated by an atom-style variable with name
```

Examples:

```
fix 1 all ave/atom 1 100 100 vx vy vz
fix 1 all ave/atom 10 20 1000 c_my_stress1
```

Description:

Use one or more per−atom vectors as inputs every few timesteps, and average them atom by atom over longer timescales. The resulting per−atom averages can be used by other output commands such as the fix ave/spatial or dump custom commands.

The group specified with the command means only atoms within the group have their averages computed. Results are set to 0.0 for atoms not in the group.

Each input value can be an atom attribute (position, velocity, force component) or can be the result of a compute or fix or the evaluation of an atom−style variable. In the latter cases, the compute, fix, or variable must produce a per−atom vector, not a global quantity or local quantity. If you wish to time−average global quantities from a compute, fix, or variable, then see the fix ave/time command.

Computes that produce per−atom vectors or arrays are those which have the word *atom* in their style name. See the doc pages for individual fixes to determine which ones produce per−atom vectors or arrays. Variables of style *atom* are the only ones that can be used with this fix since they produce per−atom vectors.

Each per−atom value of each input vector is averaged independently.

The *Nevery*, *Nrepeat*, and *Nfreq* arguments specify on what timesteps the input values will be used in order to contribute to the average. The final averaged quantities are generated on timesteps that are a multiple of *Nfreq*. The average is over *Nrepeat* quantities, computed in the preceding portion of the simulation every *Nevery* timesteps. *Nfreq* must be a multiple of *Nevery* and *Nevery* must be non−zero even if *Nrepeat* is 1. Also, the timesteps contributing to the average value cannot overlap, i.e. Nfreq > (Nrepeat−1)*Nevery is required.

For example, if Nevery=2, Nrepeat=6, and Nfreq=100, then values on timesteps 90,92,94,96,98,100 will be used to compute the final average on timestep 100. Similarly for timesteps 190,192,194,196,198,200 on timestep 200, etc.

The atom attribute values (x,y,z,vx,vy,vz,fx,fy,fz) are self−explanatory. Note that other atom attributes can be used as inputs to this fix by using the compute property/atom command and then specifying an input value from that compute.

If a value begins with "c_", a compute ID must follow which has been previously defined in the input script. If no bracketed term is appended, the per−atom vector calculated by the compute is used. If a bracketed term containing an index I is appended, the Ith column of the per−atom array calculated by the compute is used. Users can also write code for their own compute styles and add them to LAMMPS.

If a value begins with "f_", a fix ID must follow which has been previously defined in the input script. If no bracketed term is appended, the per−atom vector calculated by the fix is used. If a bracketed term containing an index I is appended, the Ith column of the per−atom array calculated by the fix is used. Note that some fixes only produce their values on certain timesteps, which must be compatible with *Nevery*, else an error will result. Users can also write code for their own fix styles and add them to LAMMPS.

If a value begins with "v_", a variable name must follow which has been previously defined in the input script as an atom−style variable Variables of style *atom* can reference thermodynamic keywords, or invoke other computes, fixes, or variables when they are evaluated, so this is a very general means of generating per−atom quantities to time average.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix. No global scalar or vector quantities are stored by this fix for access by various output commands.

This fix produces a per−atom vector or array which can be accessed by various output commands. A vector is produced if only a single quantity is averaged by this fix. If two or more quantities are averaged, then an array of values is produced. The per−atom values can only be accessed on timesteps that are multiples of *Nfreq* since that is when averaging is performed.

No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions: none

Related commands:

compute, fix ave/histo, fix ave/spatial, fix ave/time, variable,

Default: none

fix ave/correlate command

Syntax:

```
fix ID group−ID ave/correlate Nevery Nrepeat Nfreq value1 value2 ... keyword args ...
```

- ID, group−ID are documented in fix command
- ave/correlate = style name of this fix command
- Nevery = use input values every this many timesteps
- Nrepeat = # of correlation time windows to accumulate
- Nfreq = calculate tine window averages every this many timesteps
- one or more input values can be listed
- value = c_ID, c_ID[N], f_ID, f_ID[N], v_name

```
  c_ID = global scalar calculated by a compute with ID
  c_ID[I] = Ith component of global vector calculated by a compute with ID
  f_ID = global scalar calculated by a fix with ID
  f_ID[I] = Ith component of global vector calculated by a fix with ID
  v_name = global value calculated by an equal−style variable with name
```

- zero or more keyword/arg pairs may be appended
- keyword = *type* or *ave* or *start* or *prefactor* or *file* or *title1* or *title2* or *title3*

```
type arg = auto or upper or lower or auto/upper or auto/lower or full
    auto = correlate each value with itself
    upper = correlate each value with each succeeding value
    lower = correlate each value with each preceding value
    auto/upper = auto + upper
    auto/lower = auto + lower
    full = correlate each value with every other value, including itself = auto + upper + lowe
  ave args = one or running
    one = zero the correlation accumulation every Nfreq steps
    running = accumulate correlations continuously
  start args = Nstart
    Nstart = start accumulating correlations on this timestep
  prefactor args = value
    value = prefactor to scale all the correlation data by
  file arg = filename
    filename = name of file to output correlation data to
  title1 arg = string
    string = text to print as 1st line of output file
  title2 arg = string
    string = text to print as 2nd line of output file
  title3 arg = string
    string = text to print as 3rd line of output file
```

Examples:

```
fix 1 all ave/correlate 5 100 1000 c_myTemp file temp.correlate
fix 1 all ave/correlate 1 50 10000 &          c_thermo_press[1] c_thermo_press[2] c_thermo_press[3]
```

Description:

Use one or more global scalar values as inputs every few timesteps, calculate time correlations bewteen them at varying time intervals, and average the correlation data over longer timescales. The resulting correlation values can be time integrated by variables or used by other output commands such as thermo_style custom, and can also

be written to a file.

The group specified with this command is ignored. However, note that specified values may represent calculations performed by computes and fixes which store their own "group" definitions.

Each listed value can be the result of a compute or fix or the evaluation of an equal−style variable. In each case, the compute, fix, or variable must produce a global quantity, not a per−atom or local quantity. If you wish to spatial− or time−average or histogram per−atom quantities from a compute, fix, or variable, then see the fix ave/spatial, fix ave/atom, or fix ave/histo commands. If you wish to sum a per−atom quantity into a single global quantity, see the compute reduce command.

Computes that produce global quantities are those which do not have the word *atom* in their style name. Only a few fixes produce global quantities. See the doc pages for individual fixes for info on which ones produce such values. Variables of style *equal* are the only ones that can be used with this fix. Variables of style *atom* cannot be used, since they produce per−atom values.

The input values must either be all scalars. What kinds of correlations between input values are calculated is determined by the *type* keyword as discussed below.

The *Nevery*, *Nrepeat*, and *Nfreq* arguments specify on what timesteps the input values will be used to calculate correlation data. The input values are sampled every *Nevery* timesteps. The correlation data for the preceding samples is computed on timesteps that are a multiple of *Nfreq*. Consider a set of samples from some initial time up to an output timestep. The initial time could be the beginning of the simulation or the last output time; see the *ave* keyword for options. For the set of samples, the correlation value Cij is calculated as:

```
Cij(delta) = ave(Vi(t)*Vj(t+delta))
```

which is the correlation value between input values Vi and Vj, separated by time delta. Note that the second value Vj in the pair is always the one sampled at the later time. The ave() represents an average over every pair of samples in the set that are separated by time delta. The maximum delta used is of size (*Nrepeat*−1)***Nevery*. Thus the correlation between a pair of input values yields *Nrepeat* correlation datums:

```
Cij(0), Cij(Nevery), Cij(2*Nevery), ..., Cij((Nrepeat−1)*Nevery)
```

For example, if Nevery=5, Nrepeat=6, and Nfreq=100, then values on timesteps 0,5,10,15,...,100 will be used to compute the final averages on timestep 100. Six averages will be computed: Cij(0), Cij(5), Cij(10), Cij(15), Cij(20), and Cij(25). Cij(10) on timestep 100 will be the average of 19 samples, namely Vi(0)*Vj(10), Vi(5)*Vj(15), Vi(10)*V j20), Vi(15)*Vj(25), ..., Vi(85)*Vj(95), Vi(90)*Vj(100).

Nfreq must be a multiple of *Nevery*; *Nevery* and *Nrepeat* must be non−zero. Also, if the *ave* keyword is set to *one* which is the default, then *Nfreq* >= (*Nrepeat*−1)***Nevery* is required.

If a value begins with "c_", a compute ID must follow which has been previously defined in the input script. If no bracketed term is appended, the global scalar calculated by the compute is used. If a bracketed term is appended, the Ith element of the global vector calculated by the compute is used.

Note that there is a compute reduce command which can sum per−atom quantities into a global scalar or vector which can thus be accessed by fix ave/correlate. Or it can be a compute defined not in your input script, but by thermodynamic output or other fixes such as fix nvt or fix temp/rescale. See the doc pages for these commands which give the IDs of these computes. Users can also write code for their own compute styles and add them to LAMMPS.

If a value begins with "f_", a fix ID must follow which has been previously defined in the input script. If no bracketed term is appended, the global scalar calculated by the fix is used. If a bracketed term is appended, the Ith element of the global vector calculated by the fix is used.

Note that some fixes only produce their values on certain timesteps, which must be compatible with *Nevery*, else an error will result. Users can also write code for their own fix styles and add them to LAMMPS.

If a value begins with "v_", a variable name must follow which has been previously defined in the input script. Only equal−style variables can be referenced. See the variable command for details. Note that variables of style *equal* define a formula which can reference individual atom properties or thermodynamic keywords, or they can invoke other computes, fixes, or variables when they are evaluated, so this is a very general means of specifying quantities to time correlate.

Additional optional keywords also affect the operation of this fix.

The *type* keyword determines which pairs of input values are correlated with each other. For N input values Vi, for i = 1 to N, let the number of pairs = Npair. Note that the second value in the pair Vi(t)*Vj(t+delta) is always the one sampled at the later time.

- If *type* is set to *auto* then each input value is correlated with itself. I.e. Cii = Vi*Vi, for i = 1 to N, so Npair = N.
- If *type* is set to *upper* then each input value is correlated with every succeeding value. I.e. Cij = Vi*Vj, for i < j, so Npair = N*(N−1)/2.
- If *type* is set to *lower* then each input value is correlated with every preceeding value. I.e. Cij = Vi*Vj, for i > j, so Npair = N*(N−1)/2.
- If *type* is set to *auto/upper* then each input value is correlated with itself and every succeeding value. I.e. Cij = Vi*Vj, for i >= j, so Npair = N*(N+1)/2.
- If *type* is set to *auto/lower* then each input value is correlated with itself and every preceding value. I.e. Cij = Vi*Vj, for i <= j, so Npair = N*(N+1)/2.
- If *type* is set to *full* then each input value is correlated with itself and every other value. I.e. Cij = Vi*Vj, for i,j = 1,N so Npair = N^2.

The *ave* keyword determines what happens to the accumulation of correlation samples every *Nfreq* timesteps. If the *ave* setting is *one*, then the accumulation is restarted or zeroed every *Nfreq* timesteps. Thus the outputs on successive *Nfreq* timesteps are essentially independent of each other. The exception is that the Cij(0) = Vi(T)*Vj(T) value at a timestep T, where T is a multiple of *Nfreq*, contributes to the correlation output both at time T and at time T+Nfreq.

If the *ave* setting is *running*, then the accumulation is never zeroed. Thus the output of correlation data at any timestep is the average over samples accumulated every *Nevery* steps since the fix was defined. it can only be restarted by deleting the fix via the unfix command, or by re−defining the fix by re−specifying it.

The *start* keyword specifies what timestep the accumulation of correlation samples will begin on. The default is step 0. Setting it to a larger value can avoid adding non−equilibrated data to the correlation averages.

The *prefactor* keyword specifies a constant which will be used as a multiplier on the correlation data after it is averaged. It is effectively a scale factor on Vi*Vj, which can be used to account for the size of the time window or other unit conversions.

The *file* keyword allows a filename to be specified. Every *Nfreq* steps, an array of correlation data is written to the file. The number of rows is *Nrepeat*, as described above. The number of columns is the Npair+2, also as described above. Thus the file ends up to be a series of these array sections.

The *title1* and *title2* and *title3* keywords allow specification of the strings that will be printed as the first 3 lines of the output file, assuming the *file* keyword was used. LAMMPS uses default values for each of these, so they do not need to be specified.

By default, these header lines are as follows:

```
# Time-correlated data for fix ID
# TimeStep Number-of-time-windows
# Index TimeDelta Ncount valueI*valueJ valueI*valueJ ...
```

In the first line, ID is replaced with the fix-ID. The second line describes the two values that are printed at the first of each section of output. In the third line the value pairs are replaced with the appropriate fields from the fix ave/correlate command.

Let Sij = a set of time correlation data for input values I and J, namely the *Nrepeat* values:

```
Sij = Cij(0), Cij(Nevery), Cij(2*Nevery), ..., Cij(*Nrepeat-1)*Nevery)
```

As explained below, these datums are output as one column of a global array, which is effectively the correlation matrix.

The *trap* function defined for equal-style variables can be used to perform a time integration of this vector of datums, using a trapezoidal rule. This is useful for calculating various quantities which can be derived from time correlation data. If a normalization factor is needed for the time integration, it can be included in the variable formula or via the *prefactor* keyword.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix.

This fix computes a global array of values which can be accessed by various output commands. The values can only be accessed on timesteps that are multiples of *Nfreq* since that is when averaging is performed. The global array has # of rows = *Nrepeat* and # of columns = Npair+2. The first column has the time delta (in timesteps) between the pairs of input values used to calculate the correlation, as described above. The 2nd column has the number of samples contributing to the correlation average, as described above. The remaining Npair columns are for I,J pairs of the N input values, as determined by the *type* keyword, as described above.

- For *type* = *auto*, the Npair = N columns are ordered: C11, C22, ..., CNN.
- For *type* = *upper*, the Npair = N*(N-1)/2 columns are ordered: C12, C13, ..., C1N, C23, ..., C2N, C34, ..., CN-1N.
- For *type* = *lower*, the Npair = N*(N-1)/2 columns are ordered: C21, C31, C32, C41, C42, C43, ..., CN1, CN2, ..., CNN-1.
- For *type* = *auto/upper*, the Npair = N*(N+1)/2 columns are ordered: C11, C12, C13, ..., C1N, C22, C23, ..., C2N, C33, C34, ..., CN-1N, CNN.
- For *type* = *auto/lower*, the Npair = N*(N+1)/2 columns are ordered: C11, C21, C22, C31, C32, C33, C41, ..., C44, CN1, CN2, ..., CNN-1, CNN.
- For *type* = *full*, the Npair = N^2 columns are ordered: C11, C12, ..., C1N, C21, C22, ..., C2N, C31, ..., C3N, ..., CN1, ..., CNN-1, CNN.

The array values calculated by this fix are treated as "intensive". If you need to divide them by the number of atoms, you must do this in a later processing step, e.g. when using them in a variable.

No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions: none

Related commands:

compute, fix ave/time, fix ave/atom, fix ave/spatial, fix ave/histo, variable

Default: none

The option defaults are ave = one, type = auto, start = 0, no file output, title 1,2,3 = strings as described above, and prefactor = 1.0.

fix ave/histo command

Syntax:

```
fix ID group−ID ave/histo Nevery Nrepeat Nfreq lo hi Nbin value1 value2 ... keyword args ...
```

- ID, group−ID are documented in fix command
- ave/histo = style name of this fix command
- Nevery = use input values every this many timesteps
- Nrepeat = # of times to use input values for calculating histogram
- Nfreq = calculate histogram every this many timesteps
- lo,hi = lo/hi bounds within which to histogram
- Nbin = # of histogram bins
- one or more input values can be listed
- value = x, y, z, vx, vy, vz, fx, fy, fz, c_ID, c_ID[N], f_ID, f_ID[N], v_name

```
x,y,z,vx,vy,vz,fx,fy,fz = atom attribute (position, velocity, force component)
c_ID = scalar or vector calculated by a compute with ID
c_ID[I] = Ith component of vector or Ith column of array calculated by a compute with ID
f_ID = scalar or vector calculated by a fix with ID
f_ID[I] = Ith component of vector or Ith column of array calculated by a fix with ID
v_name = value(s) calculated by an equal−style or atom−style variable with name
```

- zero or more keyword/arg pairs may be appended
- keyword = *mode* or *file* or *ave* or *start* or *beyond* or *title1* or *title2* or *title3*

```
mode arg = scalar or vector
    scalar = all input values are scalars
    vector = all input values are vectors
  file arg = filename
    filename = name of file to output histogram(s) to
  ave args = one or running or window
    one = output a new average value every Nfreq steps
    running = output cumulative average of all previous Nfreq steps
    window M = output average of M most recent Nfreq steps
  start args = Nstart
    Nstart = start averaging on this timestep
  beyond arg = ignore or end or extra
    ignore = ignore values outside histogram lo/hi bounds
    end = count values outside histogram lo/hi bounds in end bins
    extra = create 2 extra bins for value outside histogram lo/hi bounds
  title1 arg = string
    string = text to print as 1st line of output file
  title2 arg = string
    string = text to print as 2nd line of output file
  title3 arg = string
    string = text to print as 3rd line of output file, only for vector mode
```

Examples:

```
fix 1 all ave/histo 100 5 1000 0.5 1.5 50 c_myTemp file temp.histo ave running
fix 1 all ave/histo 100 5 1000 −5 5 100 c_thermo_press[2] c_thermo_press[3] title1 "My output values
fix 1 all ave/histo 1 100 1000 −2.0 2.0 18 vx vy vz mode vector ave running beyond extra
```

Description:

Use one or more values as inputs every few timesteps, histogram them, and average the histogram over longer timescales. The resulting histogram can be used by other output commands, and can also be written to a file.

The group specified with this command is ignored for global and local input values. For per−atom input values, only atoms in the group contribute to the histogram. Note that regardless of the specified group, specified values may represent calculations performed by computes and fixes which store their own "group" definition.

A histogram is simply a count of the number of values that fall within a histogram bin. *Nbins* are defined, with even spacing between *lo* and *hi*. Values that fall outside the lo/hi bounds can be treated in different ways; see the discussion of the *beyond* keyword below.

Each input value can be an atom attribute (position, velocity, force component) or can be the result of a compute or fix or the evaluation of an equal−style or atom−style variable. The set of input values can be either all global, all per−atom, or all local quantities. Inputs of different kinds (e.g. global and per−atom) cannot be mixed. Atom attributes are per−atom vector values. See the doc page for individual "compute" and "fix" commands to see what kinds of quantities they generate.

The input values must either be all scalars or all vectors (or arrays), depending on the setting of the *mode* keyword.

If *mode* = vector, then the input values may either be vectors or arrays. If a global array is listed, then it is the same as if the individual columns of the array had been listed one by one. E.g. these 2 fix ave/histo commands are equivalent, since the compute com/molecule command creates a global array with 3 columns:

```
compute myCOM all com/molecule
fix 1 all ave/histo 100 1 100 c_myCOM file tmp1.com mode vector
fix 2 all ave/histo 100 1 100 c_myCOM[1] c_myCOM[2] c_myCOM[3] file tmp2.com mode vector
```

The output of this command is a single histogram for all input values combined together, not one histogram per input value. See below for details on the format of the output of this fix.

The *Nevery*, *Nrepeat*, and *Nfreq* arguments specify on what timesteps the input values will be used in order to contribute to the histogram. The final histogram is generated on timesteps that are multiple of *Nfreq*. It is averaged over *Nrepeat* histograms, computed in the preceding portion of the simulation every *Nevery* timesteps. *Nfreq* must be a multiple of *Nevery* and *Nevery* must be non−zero even if *Nrepeat* is 1. Also, the timesteps contributing to the histogram cannot overlap, i.e. Nfreq > (Nrepeat−1)*Nevery is required.

For example, if Nevery=2, Nrepeat=6, and Nfreq=100, then input values on timesteps 90,92,94,96,98,100 will be used to compute the final histogram on timestep 100. Similarly for timesteps 190,192,194,196,198,200 on timestep 200, etc. If Nrepeat=1 and Nfreq = 100, then no time averaging of the histogram is done; a histogram is simply generated on timesteps 100,200,etc.

The atom attribute values (x,y,z,vx,vy,vz,fx,fy,fz) are self−explanatory. Note that other atom attributes can be used as inputs to this fix by using the compute property/atom command and then specifying an input value from that compute.

If a value begins with "c_", a compute ID must follow which has been previously defined in the input script. If *mode* = scalar, then if no bracketed term is appended, the global scalar calculated by the compute is used. If a bracketed term is appended, the Ith element of the global vector calculated by the compute is used. If *mode* = vector, then if no bracketed term is appended, the global or per−atom or local vector calculated by the compute is used. Or if the compute calculates an array, all of the columns of the array are used as if they had been specified as individual vectors (see description above). If a bracketed term is appended, the Ith column of the global or per−atom or local array calculated by the compute is used.

Note that there is a compute reduce command which can sum per−atom quantities into a global scalar or vector which can thus be accessed by fix ave/histo. Or it can be a compute defined not in your input script, but by thermodynamic output or other fixes such as fix nvt or fix temp/rescale. See the doc pages for these commands which give the IDs of these computes. Users can also write code for their own compute styles and add them to LAMMPS.

If a value begins with "f_", a fix ID must follow which has been previously defined in the input script. If *mode* = scalar, then if no bracketed term is appended, the global scalar calculated by the fix is used. If a bracketed term is appended, the Ith element of the global vector calculated by the fix is used. If *mode* = vector, then if no bracketed term is appended, the global or per−atom or local vector calculated by the fix is used. Or if the fix calculates an array, all of the columns of the array are used as if they had been specified as individual vectors (see description above). If a bracketed term is appended, the Ith column of the global or per−atom or local array calculated by the fix is used.

Note that some fixes only produce their values on certain timesteps, which must be compatible with *Nevery*, else an error will result. Users can also write code for their own fix styles and add them to LAMMPS.

If a value begins with "v_", a variable name must follow which has been previously defined in the input script. If *mode* = scalar, then only equal−style variables can be used, which produce a global value. If *mode* = vector, then only atom−style variables can be used, which produce a per−atom vector. See the variable command for details. Note that variables of style *equal* and *atom* define a formula which can reference individual atom properties or thermodynamic keywords, or they can invoke other computes, fixes, or variables when they are evaluated, so this is a very general means of specifying quantities to histogram.

Additional optional keywords also affect the operation of this fix.

If the *mode* keyword is set to *scalar*, then all input values must be global scalars, or elements of global vectors. If the *mode* keyword is set to *vector*, then all input values must be global or per−atom or local vectors, or columns of global or per−atom or local arrays.

The *beyond* keyword determines how input values that fall outside the *lo* to *hi* bounds are treated. Values such that *lo* <= value <= *hi* are assigned to one bin. Values on a bin boundary are assigned to the lower of the 2 bins. If *beyond* is set to *ignore* then values < *lo* and values > *hi* are ignored, i.e. they are not binned. If *beyond* is set to *end* then values < *lo* are counted in the first bin and values > *hi* are counted in the last bin. If *beyond* is set to *extend* then two extra bins are created, so that there are Nbins+2 total bins. Values < *lo* are counted in the first bin and values > *hi* are counted in the last bin (Nbins+1). Values between *lo* and *hi* (inclusive) are counted in bins 2 thru Nbins+1. The "coordinate" stored and printed for these two extra bins is *lo* and *hi*.

The *ave* keyword determines how the histogram produced every *Nfreq* steps are averaged with histograms produced on previous steps that were multiples of *Nfreq*, before they are accessed by another output command or written to a file.

If the *ave* setting is *one*, then the histograms produced on timesteps that are multiples of *Nfreq* are independent of each other; they are output as−is without further averaging.

If the *ave* setting is *running*, then the histograms produced on timesteps that are multiples of *Nfreq* are summed and averaged in a cumulative sense before being output. Each bin value in the histogram is thus the average of the bin value produced on that timestep with all preceding values for the same bin. This running average begins when the fix is defined; it can only be restarted by deleting the fix via the unfix command, or by re−defining the fix by re−specifying it.

If the *ave* setting is *window*, then the histograms produced on timesteps that are multiples of *Nfreq* are summed

within a moving "window" of time, so that the last M histograms are used to produce the output. E.g. if M = 3 and Nfreq = 1000, then the output on step 10000 will be the combined histogram of the individual histograms on steps 8000,9000,10000. Outputs on early steps will be sums over less than M histograms if they are not available.

The *start* keyword specifies what timestep histogramming will begin on. The default is step 0. Often input values can be 0.0 at time 0, so setting *start* to a larger value can avoid including a 0.0 in a running or windowed histogram.

The *file* keyword allows a filename to be specified. Every *Nfreq* steps, one histogram is written to the file. This includes a leading line that contains the timestep, number of bins, the total count of values contributing to the histogram, the count of values that were not histogrammed (see the *beyond* keyword), the minimum value encountered, and the maximum value encountered. The min/max values include values that were not histogrammed. Following the leading line, one line per bin is written into the file. Each line contains the bin #, the coordinate for the center of the bin (between *lo* and *hi*), the count of values in the bin, and the normalized count. The normalized count is the bin count divided by the total count (not including values not histogrammed), so that the normalized values sum to 1.0 across all bins.

The *title1* and *title2* and *title3* keywords allow specification of the strings that will be printed as the first 3 lines of the output file, assuming the *file* keyword was used. LAMMPS uses default values for each of these, so they do not need to be specified.

By default, these header lines are as follows:

```
# Histogram for fix ID
# TimeStep Number-of-bins Total-counts Missing-counts Min-value Max-value
# Bin Coord Count Count/Total
```

In the first line, ID is replaced with the fix−ID. The second line describes the six values that are printed at the first of each section of output. The third describes the 4 values printed for each bin in the histogram.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix.

This fix produces a global vector and global array which can be accessed by various output commands. The values can only be accessed on timesteps that are multiples of *Nfreq* since that is when a histogram is generated. The global vector has 4 values:

- 1 = total counts in the histogram
- 2 = values that were not histogrammed (see *beyond* keyword)
- 3 = min value of all input values, including ones not histogrammed
- 4 = max value of all input values, including ones not histogrammed

The global array has # of rows = Nbins and # of columns = 3. The first column has the bin coordinate, the 2nd column has the count of values in that histogram bin, and the 3rd column has the bin count divided by the total count (not including missing counts), so that the values in the 3rd column sum to 1.0.

The vector and array values calculated by this fix are all treated as "intensive". If this is not the case, e.g. due to histogramming per−atom input values, then you will need to account for that when interpreting the values produced by this fix.

No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions: none

Related commands:

compute, fix ave/atom, fix ave/spatial, fix ave/time, variable, fix ave/correlate,

Default: none

The option defaults are mode = scalar, ave = one, start = 0, no file output, beyond = ignore, and title 1,2,3 = strings as described above.

fix ave/spatial command

Syntax:

```
fix ID group−ID ave/spatial Nevery Nrepeat Nfreq dim origin delta ... value1 value2 ... keyword args
```

- ID, group−ID are documented in fix command
- ave/spatial = style name of this fix command
- Nevery = use input values every this many timesteps
- Nrepeat = # of times to use input values for calculating averages
- Nfreq = calculate averages every this many timesteps
- dim, origin, delta can be repeated 1, 2, or 3 times for 1d, 2d, or 3d bins

```
dim = x or y or z
origin = lower or center or upper or coordinate value (distance units)
delta = thickness of spatial bins in dim (distance units)
```

- one or more input values can be listed
- value = vx, vy, vz, fx, fy, fz, density/mass, density/number, c_ID, c_ID[I], f_ID, f_ID[I], v_name

```
vx,vy,vz,fx,fy,fz = atom attribute (velocity, force component)
density/number, density/mass = number or mass density
c_ID = per−atom vector calculated by a compute with ID
c_ID[I] = Ith column of per−atom array calculated by a compute with ID
f_ID = per−atom vector calculated by a fix with ID
f_ID[I] = Ith column of per−atom array calculated by a fix with ID
v_name = per−atom vector calculated by an atom−style variable with name
```

- zero or more keyword/arg pairs may be appended
- keyword = *norm* or *units* or *file* or *ave* or *title1* or *title2* or *title3*

```
units arg = box or lattice or reduced
  norm arg = all or sample
  region arg = region−ID
    region−ID = ID of region atoms must be in to contribute to spatial averaging
  ave args = one or running or window M
    one = output new average value every Nfreq steps
    running = output cumulative average of all previous Nfreq steps
    window M = output average of M most recent Nfreq steps
  file arg = filename
    filename = file to write results to
  title1 arg = string
    string = text to print as 1st line of output file
  title2 arg = string
    string = text to print as 2nd line of output file
  title3 arg = string
    string = text to print as 3rd line of output file
```

Examples:

```
fix 1 all ave/spatial 10000 1 10000 z lower 0.02 c_myCentro units reduced &                    tit
fix 1 flow ave/spatial 100 10 1000 y 0.0 1.0 vx vz norm sample file vel.profile
fix 1 flow ave/spatial 100 5 1000 z lower 1.0 y 0.0 2.5 density/mass ave running
```

Description:

Use one or more per−atom vectors as inputs every few timesteps, bin their values spatially into 1d, 2d, or 3d bins

based on current atom coordinates, and average the bin values over longer timescales. The resulting bin averages can be used by other output commands such as thermo_style custom, and can also be written to a file.

The group specified with the command means only atoms within the group contribute to bin averages. If the *region* keyword is used, the atom must be in both the group and the specified geometric region in order to contribute to bin averages.

Each listed value can be an atom attribute (position, velocity, force component), a mass or number density, or the result of a compute or fix or the evaluation of an atom−style variable. In the latter cases, the compute, fix, or variable must produce a per−atom quantity, not a global quantity. If you wish to time−average global quantities from a compute, fix, or variable, then see the fix ave/time command.

Computes that produce per−atom quantities are those which have the word *atom* in their style name. See the doc pages for individual fixes to determine which ones produce per−atom quantities. Variables of style *atom* are the only ones that can be used with this fix since all other styles of variable produce global quantities.

The per−atom values of each input vector are binned and averaged independently of the per−atom values in other input vectors.

The size and dimensionality of the bins (1d = layers or slabs, 2d = pencils, 3d = boxes) are determined by the *dim*, *origin*, and *delta* settings and how many times they are specified (1, 2, or 3). See details below.

IMPORTANT NOTE: This fix works by creating an array of size Nbins by Nvalues on each processor. Nbins is the total number of bins; Nvalues is the number of input values specified. Each processor loops over its atoms, tallying its values to the appropriate bin. Then the entire array is summed across all processors. This means that using a large number of bins (easy to do for 2d or 3d bins) will incur an overhead in memory and computational cost (summing across processors), so be careful to use reasonable numbers of bins.

The *Nevery*, *Nrepeat*, and *Nfreq* arguments specify on what timesteps the input values will be used to bin them and contribute to the average. The final averaged quantities are generated on timesteps that are a multiples of *Nfreq*. The average is over *Nrepeat* quantities, computed in the preceding portion of the simulation every *Nevery* timesteps. *Nfreq* must be a multiple of *Nevery* and *Nevery* must be non−zero even if *Nrepeat* is 1. Also, the timesteps contributing to the average value cannot overlap, i.e. Nfreq > (Nrepeat−1)*Nevery is required.

For example, if Nevery=2, Nrepeat=6, and Nfreq=100, then values on timesteps 90,92,94,96,98,100 will be used to compute the final average on timestep 100. Similarly for timesteps 190,192,194,196,198,200 on timestep 200, etc. If Nrepeat=1 and Nfreq = 100, then no time averaging is done; values are simply generated on timesteps 100,200,etc.

Each per−atom property is also averaged over atoms in each bin. Bins can be 1d layers or slabs, 2d pencils, or 3d boxes. This depends on how many times (1, 2, or 3) the *dim*, *origin*, and *delta* settings are specified in the fix ave/spatial command. For 2d or 3d bins, there is no restriction on specifying dim = x before dim = y, or dim = y before dim = z. Bins in a particular *dim* have a bin size in that dimension given by *delta*. Every Nfreq steps, when averaging is being performed and the per−atom property is calculated for the first time, the number of bins and the bin sizes and boundaries are computed. Thus if the simulation box changes size during a simulation, the number of bins and their boundaries may also change. In each dimension, bins are defined relative to a specified *origin*, which may be the lower/upper edge of the simulation box (in *dim*) or its center point, or a specified coordinate value. Starting at the origin, sufficient bins are created in both directions to completely cover the box. On subsequent timesteps every atom is mapped to one of the bins. Atoms beyond the lowermost/uppermost bin in a dimension are counted in the first/last bin in that dimension.

For orthogonal simulation boxes, the bins are also layers, pencils, or boxes aligned with the xyz coordinate axes. For triclinic (non−orthogonal) simulation boxes, the bins are so that they are parallel to the tilted faces of the simulation box. See this section of the manual for a discussion of the geometry of triclinic boxes in LAMMPS. As described there, a tilted simulation box has edge vectors a,b,c. In that nomenclature, bins in the x dimension have faces with normals in the "b" cross "c" direction. Bins in y have faces normal to the "a" cross "c" direction. And bins in z have faces normal to the "a" cross "b" direction. Note that in order to define the size and position of these bins in an unambiguous fashion, the *units* option must be set to *reduced* when using a triclinic simulation box, as noted below.

The atom attribute values (vx,vy,vz,fx,fy,fz) are self−explanatory. Note that other atom attributes (including atom postitions x,y,z) can be used as inputs to this fix by using the compute property/atom command and then specifying an input value from that compute.

The *density/number* value means the number density is computed in each bin, i.e. a weighting of 1 for each atom. The *density/mass* value means the mass density is computed in each bind, i.e. each atom is weighted by its mass. The resulting density is normalized by the volume of the bin so that units of number/volume or density are output. See the units command doc page for the definition of density for each choice of units, e.g. gram/cm^3.

If a value begins with "c_", a compute ID must follow which has been previously defined in the input script. If no bracketed integer is appended, the per−atom vector calculated by the compute is used. If a bracketed interger is appended, the Ith column of the per−atom array calculated by the compute is used. Users can also write code for their own compute styles and add them to LAMMPS.

If a value begins with "f_", a fix ID must follow which has been previously defined in the input script. If no bracketed integer is appended, the per−atom vector calculated by the fix is used. If a bracketed integer is appended, the Ith column of the per−atom array calculated by the fix is used. Note that some fixes only produce their values on certain timesteps, which must be compatible with *Nevery*, else an error results. Users can also write code for their own fix styles and add them to LAMMPS.

If a value begins with "v_", a variable name must follow which has been previously defined in the input script. Variables of style *atom* can reference thermodynamic keywords and various per−atom attributes, or invoke other computes, fixes, or variables when they are evaluated, so this is a very general means of generating per−atom quantities to spatially average.

Additional optional keywords also affect the operation of this fix.

The *units* keyword determines the meaning of the distance units used for the bin size *delta* and for *origin* if it is a coordinate value. For orthogonal simulation boxes, any of the 3 options may be used. For non−orthogonal (triclinic) simulation boxes, only the *reduced* option may be used.

A *box* value selects standard distance units as defined by the units command, e.g. Angstroms for units = real or metal. A *lattice* value means the distance units are in lattice spacings. The lattice command must have been previously used to define the lattice spacing. A *reduced* value means normalized unitless values between 0 and 1, which represent the lower and upper faces of the simulation box respectively. Thus an *origin* value of 0.5 means the center of the box in any dimension. A *delta* value of 0.1 means 10 bins span the box in that dimension.

Consider a non−orthogonal box, with bins that are 1d layers or slabs in the x dimension. No matter how the box is tilted, an *origin* of 0.0 means start layers at the lower "b" cross "c" plane of the simulation box and an *origin* of 1.0 means to start layers at the upper "b" cross "c" face of the box. A *delta* value of 0.1 means there will be 10 layers from 0.0 to 1.0, regardless of the current size or shape of the simulation box.

The *norm* keyword affects how averaging is done for the output produced every *Nfreq* timesteps. For an *all* setting, a bin quantity is summed over all atoms in all *Nrepeat* samples, as is the count of atoms in the bin. The printed value for the bin is Total−quantity / Total−count. In other words it is an average over the entire *Nfreq* timescale.

For a *sample* setting, the bin quantity is summed over atoms for only a single sample, as is the count, and a "average sample value" is computed, i.e. Sample−quantity / Sample−count. The printed value for the bin is the average of the *Nrepeat* "average sample values", In other words it is an average of an average.

The *ave* keyword determines how the bin values produced every *Nfreq* steps are averaged with bin values produced on previous steps that were multiples of *Nfreq*, before they are accessed by another output command or written to a file.

If the *ave* setting is *one*, then the bin values produced on timesteps that are multiples of *Nfreq* are independent of each other; they are output as−is without further averaging.

If the *ave* setting is *running*, then the bin values produced on timesteps that are multiples of *Nfreq* are summed and averaged in a cumulative sense before being output. Each output bin value is thus the average of the bin value produced on that timestep with all preceding values for the same bin. This running average begins when the fix is defined; it can only be restarted by deleting the fix via the unfix command, or re−defining the fix by re−specifying it.

If the *ave* setting is *window*, then the bin values produced on timesteps that are multiples of *Nfreq* are summed and averaged within a moving "window" of time, so that the last M values for the same bin are used to produce the output. E.g. if M = 3 and Nfreq = 1000, then the output on step 10000 will be the average of the individual bin values on steps 8000,9000,10000. Outputs on early steps will average over less than M values if they are not available.

The *file* keyword allows a filename to be specified. Every *Nfreq* timesteps, a section of bin info will be written to a text file in the following format. A line with the timestep and number of bin is written. Then one line per bin is written, containing the bin ID (1−N), the coordinate of the center of the bin, the number of atoms in the bin, and one or more calculated values. The number of values in each line corresponds to the number of values specified in the fix ave/spatial command. The number of atoms and the value(s) are average quantities. If the value of the *units* keyword is *box* or *lattice*, the "coord" is printed in box units. If the value of the *units* keyword is *reduced*, the "coord" is printed in reduced units (0−1).

The *title1* and *title2* and *title3* keywords allow specification of the strings that will be printed as the first 3 lines of the output file, assuming the *file* keyword was used. LAMMPS uses default values for each of these, so they do not need to be specified.

By default, these header lines are as follows:

```
# Spatial−averaged data for fix ID and group name
# Timestep Number−of−bins
# Bin Coord1 Coord2 Coord3 Count value1 value2 ...
```

In the first line, ID and name are replaced with the fix−ID and group name. The second line describes the two values that are printed at the first of each section of output. In the third line the values are replaced with the appropriate fields from the fix ave/spatial command. The Coord2 and Coord3 entries in the third line only appear for 2d and 3d bins respectively. For 1d bins, the word Coord1 is replaced by just Coord.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix.

This fix computes a global array of values which can be accessed by various output commands. The values can only be accessed on timesteps that are multiples of *Nfreq* since that is when averaging is performed. The global array has # of rows = Nbins and # of columns = Ndim+1+Nvalues, where Ndim = 1,2,3 for 1d,2d,3d bins. The first 1 or 2 or 3 columns have the bin coordinates (center of the bin) in the appropriate dimensions, the next column has the count of atoms in that bin, and the remaining columns are the Nvalue quantities. When the array is accessed with an I that exceeds the current number of bins, than a 0.0 is returned by the fix instead of an error, since the number of bins can vary as a simulation runs, depending on the simulation box size. 2d or 3d bins are ordered so that the last dimension(s) vary fastest. The array values calculated by this fix are "intensive", since they are already normalized by the count of atoms in each bin.

No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions:

When the *ave* keyword is set to *running* or *window* then the number of bins must remain the same during the simulation, so that the appropriate averaging can be done. This will be the case if the simulation box size doesn't change or if the *units* keyword is set to *reduced*.

Related commands:

compute, fix ave/atom, fix ave/histo, fix ave/time, variable, fix ave/correlate,

Default:

The option defaults are units = lattice, norm = all, no file output, and ave = one, title 1,2,3 = strings as described above.

fix ave/time command

Syntax:

```
fix ID group−ID ave/time Nevery Nrepeat Nfreq value1 value2 ... keyword args ...
```

- ID, group−ID are documented in fix command
- ave/time = style name of this fix command
- Nevery = use input values every this many timesteps
- Nrepeat = # of times to use input values for calculating averages
- Nfreq = calculate averages every this many timesteps
- one or more input values can be listed
- value = c_ID, c_ID[N], f_ID, f_ID[N], v_name

```
  c_ID = global scalar or vector calculated by a compute with ID
  c_ID[I] = Ith component of global vector or Ith column of global array calculated by a compu
  f_ID = global scalar or vector calculated by a fix with ID
  f_ID[I] = Ith component of global vector or Ith column of global array calculated by a fix v
  v_name = global value calculated by an equal−style variable with name
```

- zero or more keyword/arg pairs may be appended
- keyword = *mode* or *file* or *ave* or *start* or *off* or *title1* or *title2* or *title3*

```
mode arg = scalar or vector
    scalar = all input values are global scalars
    vector = all input values are global vectors or global arrays
  ave args = one or running or window M
    one = output a new average value every Nfreq steps
    running = output cummulative average of all previous Nfreq steps
    window M = output average of M most recent Nfreq steps
  start args = Nstart
    Nstart = start averaging on this timestep
  off arg = M = do not average this value
    M = value # from 1 to Nvalues
  file arg = filename
    filename = name of file to output time averages to
  title1 arg = string
    string = text to print as 1st line of output file
  title2 arg = string
    string = text to print as 2nd line of output file
  title3 arg = string
    string = text to print as 3rd line of output file, only for vector mode
```

Examples:

```
fix 1 all ave/time 100 5 1000 c_myTemp c_thermo_temp file temp.profile
fix 1 all ave/time 100 5 1000 c_thermo_press[2] ave window 20 &                                 title1
fix 1 all ave/time 1 100 1000 f_indent f_indent[1] file temp.indent off 1
```

Description:

Use one or more global values as inputs every few timesteps, and average them over longer timescales. The resulting averages can be used by other output commands such as thermo_style custom, and can also be written to a file. Note that if no time averaging is done, this command can be used as a convenient way to simply output one or more global values to a file.

The group specified with this command is ignored. However, note that specified values may represent calculations performed by computes and fixes which store their own "group" definitions.

Each listed value can be the result of a compute or fix or the evaluation of an equal−style variable. In each case, the compute, fix, or variable must produce a global quantity, not a per−atom or local quantity. If you wish to spatial− or time−average or histogram per−atom quantities from a compute, fix, or variable, then see the fix ave/spatial, fix ave/atom, or fix ave/histo commands. If you wish to sum a per−atom quantity into a single global quantity, see the compute reduce command.

Computes that produce global quantities are those which do not have the word *atom* in their style name. Only a few fixes produce global quantities. See the doc pages for individual fixes for info on which ones produce such values. Variables of style *equal* are the only ones that can be used with this fix. Variables of style *atom* cannot be used, since they produce per−atom values.

The input values must either be all scalars or all vectors (or arrays), depending on the setting of the *mode* keyword. In both cases, the averaging is performed independently on each input value. I.e. each input scalar is averaged independently and each element of each input vector (or array) is averaged independently.

If *mode* = vector, then the input values may either be vectors or arrays and all must be the same "length", which is the length of the vector or number of rows in the array. If a global array is listed, then it is the same as if the individual columns of the array had been listed one by one. E.g. these 2 fix ave/time commands are equivalent, since the compute rdf command creates, in this case, a global array with 3 columns, each of length 50:

```
compute myRDF all rdf 50 1 2
fix 1 all ave/time 100 1 100 c_myRDF file tmp1.rdf mode vector
fix 2 all ave/time 100 1 100 c_myRDF[1] c_myRDF[2] c_myRDF[3] file tmp2.rdf mode vector
```

The *Nevery*, *Nrepeat*, and *Nfreq* arguments specify on what timesteps the input values will be used in order to contribute to the average. The final averaged quantities are generated on timesteps that are a mlutiple of *Nfreq*. The average is over *Nrepeat* quantities, computed in the preceding portion of the simulation every *Nevery* timesteps. *Nfreq* must be a multiple of *Nevery* and *Nevery* must be non−zero even if *Nrepeat* is 1. Also, the timesteps contributing to the average value cannot overlap, i.e. Nfreq > (Nrepeat−1)*Nevery is required.

For example, if Nevery=2, Nrepeat=6, and Nfreq=100, then values on timesteps 90,92,94,96,98,100 will be used to compute the final average on timestep 100. Similarly for timesteps 190,192,194,196,198,200 on timestep 200, etc. If Nrepeat=1 and Nfreq = 100, then no time averaging is done; values are simply generated on timesteps 100,200,etc.

If a value begins with "c_", a compute ID must follow which has been previously defined in the input script. If *mode* = scalar, then if no bracketed term is appended, the global scalar calculated by the compute is used. If a bracketed term is appended, the Ith element of the global vector calculated by the compute is used. If *mode* = vector, then if no bracketed term is appended, the global vector calculated by the compute is used. Or if the compute calculates an array, all of the columns of the global array are used as if they had been specified as individual vectors (see description above). If a bracketed term is appended, the Ith column of the global array calculated by the compute is used.

Note that there is a compute reduce command which can sum per−atom quantities into a global scalar or vector which can thus be accessed by fix ave/time. Or it can be a compute defined not in your input script, but by thermodynamic output or other fixes such as fix nvt or fix temp/rescale. See the doc pages for these commands which give the IDs of these computes. Users can also write code for their own compute styles and add them to LAMMPS.

If a value begins with "f_", a fix ID must follow which has been previously defined in the input script. If *mode* = scalar, then if no bracketed term is appended, the global scalar calculated by the fix is used. If a bracketed term is appended, the Ith element of the global vector calculated by the fix is used. If *mode* = vector, then if no bracketed term is appended, the global vector calculated by the fix is used. Or if the fix calculates an array, all of the columns of the global array are used as if they had been specified as individual vectors (see description above). If a bracketed term is appended, the Ith column of the global array calculated by the fix is used.

Note that some fixes only produce their values on certain timesteps, which must be compatible with *Nevery*, else an error will result. Users can also write code for their own fix styles and add them to LAMMPS.

If a value begins with "v_", a variable name must follow which has been previously defined in the input script. Variables can only be used as input for *mode* = scalar. Only equal−style variables can be referenced. See the variable command for details. Note that variables of style *equal* define a formula which can reference individual atom properties or thermodynamic keywords, or they can invoke other computes, fixes, or variables when they are evaluated, so this is a very general means of specifying quantities to time average.

Additional optional keywords also affect the operation of this fix.

If the *mode* keyword is set to *scalar*, then all input values must be global scalars, or elements of global vectors. If the *mode* keyword is set to *vector*, then all input values must be global vectors, or columns of global arrays. They can also be global arrays, which are converted into a series of global vectors (one per column), as explained above.

The *ave* keyword determines how the values produced every *Nfreq* steps are averaged with values produced on previous steps that were multiples of *Nfreq*, before they are accessed by another output command or written to a file.

If the *ave* setting is *one*, then the values produced on timesteps that are multiples of *Nfreq* are independent of each other; they are output as−is without further averaging.

If the *ave* setting is *running*, then the values produced on timesteps that are multiples of *Nfreq* are summed and averaged in a cummulative sense before being output. Each output value is thus the average of the value produced on that timestep with all preceding values. This running average begins when the fix is defined; it can only be restarted by deleting the fix via the unfix command, or by re−defining the fix by re−specifying it.

If the *ave* setting is *window*, then the values produced on timesteps that are multiples of *Nfreq* are summed and averaged within a moving "window" of time, so that the last M values are used to produce the output. E.g. if M = 3 and Nfreq = 1000, then the output on step 10000 will be the average of the individual values on steps 8000,9000,10000. Outputs on early steps will average over less than M values if they are not available.

The *start* keyword specifies what timestep averaging will begin on. The default is step 0. Often input values can be 0.0 at time 0, so setting *start* to a larger value can avoid including a 0.0 in a running or windowed average.

The *off* keyword can be used to flag any of the input values. If a value is flagged, it will not be time averaged. Instead the most recent input value will always be stored and output. This is useful if one of more of the inputs produced by a compute or fix or variable are effectively constant or are simply current values. E.g. they are being written to a file with other time−averaged values for purposes of creating well−formatted output.

The *file* keyword allows a filename to be specified. Every *Nfreq* steps, one quantity or vector of quantities is written to the file for each input value specified in the fix ave/time command. For *mode* = scalar, this means a single line is written each time output is performed. Thus the file ends up to be a series of lines, i.e. one column of numbers for each input value. For *mode* = vector, an array of numbers is written each time output is performed.

The number of rows is the length of the input vectors, and the number of columns is the number of values. Thus the file ends up to be a series of these array sections.

The *title1* and *title2* and *title3* keywords allow specification of the strings that will be printed as the first 2 or 3 lines of the output file, assuming the *file* keyword was used. LAMMPS uses default values for each of these, so they do not need to be specified.

By default, these header lines are as follows for *mode* = scalar:

```
# Time-averaged data for fix ID
# TimeStep value1 value2 ...
```

In the first line, ID is replaced with the fix−ID. In the second line the values are replaced with the appropriate fields from the fix ave/time command. There is no third line in the header of the file, so the *title3* setting is ignored when *mode* = scalar.

By default, these header lines are as follows for *mode* = vector:

```
# Time-averaged data for fix ID
# TimeStep Number-of-rows
# Row value1 value2 ...
```

In the first line, ID is replaced with the fix−ID. The second line describes the two values that are printed at the first of each section of output. In the third line the values are replaced with the appropriate fields from the fix ave/time command.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix.

This fix produces a global scalar or global vector or global array which can be accessed by various output commands. The values can only be accessed on timesteps that are multiples of *Nfreq* since that is when averaging is performed.

A scalar is produced if only a single input value is averaged and *mode* = scalar. A vector is produced if multiple input values are averaged for *mode* = scalar, or a single input value for *mode* = vector. In the first case, the length of the vector is the number of inputs. In the second case, the length of the vector is the same as the length of the input vector. An array is produced if multiple input values are averaged and *mode* = vector. The global array has # of rows = length of the input vectors and # of columns = number of inputs.

If the fix prouduces a scalar or vector, then the scalar and each element of the vector can be either "intensive" or "extensive". If the fix produces an array, then all elements in the array must be the same, either "intensive" or "extensive". If a compute or fix provides the value being time averaged, then the compute or fix determines whether the value is intensive or extensive; see the doc page for that compute or fix for further info. Values produced by a variable are treated as intensive.

No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions: none

Related commands:

compute, fix ave/atom, fix ave/spatial, fix ave/histo, variable, fix ave/correlate,

Default: none

The option defaults are mode = scalar, ave = one, start = 0, no file output, title 1,2,3 = strings as described above, and no off settings for any input values.

fix aveforce command

fix aveforce/cuda command

Syntax:

```
fix ID group-ID aveforce fx fy fz keyword value ...
```

- ID, group−ID are documented in fix command
- aveforce = style name of this fix command
- fx,fy,fz = force component values (force units)
- any of fx,fy,fz can be a variable (see below)
- zero or more keyword/value pairs may be appended to args
- keyword = *region*

```
region value = region-ID
    region-ID = ID of region atoms must be in to have added force
```

Examples:

```
fix pressdown topwall aveforce 0.0 -1.0 0.0
fix 2 bottomwall aveforce NULL -1.0 0.0 region top
fix 2 bottomwall aveforce NULL -1.0 v_oscillate region top
```

Description:

Apply an additional external force to a group of atoms in such a way that every atom experiences the same force. This is useful for pushing on wall or boundary atoms so that the structure of the wall does not change over time.

The existing force is averaged for the group of atoms, component by component. The actual force on each atom is then set to the average value plus the component specified in this command. This means each atom in the group receives the same force.

Any of the fx,fy,fz values can be specified as NULL which means the force in that dimension is not changed. Note that this is not the same as specifying a 0.0 value, since that sets all forces to the same average value without adding in any additional force.

Any of the 3 quantities defining the force components can be specified as an equal−style variable, namely *fx*, *fy*, *fz*. If the value is a variable, it should be specified as v_name, where name is the variable name. In this case, the variable will be evaluated each timestep, and its value used to determine the average force.

Equal−style variables can specify formulas with various mathematical functions, and include thermo_style command keywords for the simulation box parameters and timestep and elapsed time. Thus it is easy to specify a time−dependent average force.

If the *region* keyword is used, the atom must also be in the specified geometric region in order to have force added to it.

Styles with a *cuda* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in this section of the manual. The accelerated styles take the same arguments and should produce the same results, except for round−off and

precision issues.

These accelerated styles are part of the "user−cuda" package. They are only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the −suffix command−line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See this section of the manual for more instructions on how to use the accelerated styles effectively.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix.

This fix computes a global 3−vector of forces, which can be accessed by various output commands. This is the total force on the group of atoms before the forces on individual atoms are changed by the fix. The vector values calculated by this fix are "extensive".

No parameter of this fix can be used with the *start/stop* keywords of the run command.

The forces due to this fix are imposed during an energy minimization, invoked by the minimize command. You should not specify force components with a variable that has time−dependence for use with a minimizer, since the minimizer increments the timestep as the iteration count during the minimization.

Restrictions: none

Related commands:

fix setforce, fix addforce

Default: none

fix bond/break command

Syntax:

```
fix ID group−ID bond/break Nevery bondtype Rmax keyword values ...
```

- ID, group−ID are documented in fix command
- bond/break = style name of this fix command
- Nevery = attempt bond breaking every this many steps
- bondtype = type of bonds to break
- Rmax = bond longer than Rmax can break (distance units)
- zero or more keyword/value pairs may be appended to args
- keyword = *prob*

```
  prob values = fraction seed
      fraction = break a bond with this probability if otherwise eligible
      seed = random number seed (positive integer)
```

Examples:

```
fix 5 all bond/break 10 2 1.2
fix 5 polymer bond/break 1 1 2.0 prob 0.5 49829
```

Description:

Break bonds between pairs of atoms as a simulation runs according to specified criteria. This can be used to model the dissolution of a polymer network due to stretching of the simulation box or other deformations. In this context, a bond means an interaction between a pair of atoms computed by the bond_style command. Once the bond is broken it will be permanently deleted. This is different than a pairwise bond−order potential such as Tersoff or AIREBO which infers bonds and many−body interactions based on the current geometry of a small cluster of atoms and effectively creates and destroys bonds from timestep to timestep as atoms move.

A check for possible bond breakage is performed every *Nevery* timesteps. If two bonded atoms I,J are further than a distance *Rmax* of each other, if the bond is of type *bondtype*, and if both I and J are in the specified fix group, then I,J is labeled as a "possible" bond to break.

If several bonds involving an atom are stretched, it may have multiple possible bonds to break. Every atom checks its list of possible bonds to break and labels the longest such bond as its "sole" bond to break. After this is done, if atom I is bonded to atom J in its sole bond, and atom J is bonded to atom I in its sole bond, then the I,J bond is "eligible" to be broken.

Note that these rules mean an atom will only be part of at most one broken bond on a given timestep. It also means that if atom I chooses atom J as its sole partner, but atom J chooses atom K is its sole partner (due to Rjk > Rij), then this means atom I will not be part of a broken bond on this timestep, even if it has other possible bond partners.

The *prob* keyword can effect whether an eligible bond is actually broken. The *fraction* setting must be a value between 0.0 and 1.0. A uniform random number between 0.0 and 1.0 is generated and the eligible bond is only broken if the random number < fraction.

When a bond is broken, data structures within LAMMPS that store bond topology are updated to reflect the breakage. This can also affect subsequent computation of pairwise interactions involving the atoms in the bond. See the Restriction section below for additional information.

Computationally, each timestep this fix operates, it loops over bond lists and computes distances between pairs of bonded atoms in the list. It also communicates between neighboring processors to coordinate which bonds are broken. Thus it will increase the cost of a timestep. Thus you should be cautious about invoking this fix too frequently.

You can dump out snapshots of the current bond topology via the dump local command.

IMPORTANT NOTE: Breaking a bond typically alters the energy of a system. You should be careful not to choose bond breaking criteria that induce a dramatic change in energy. For example, if you define a very stiff harmonic bond and break it when 2 atoms are separated by a distance far from the equilibribum bond length, then the 2 atoms will be dramatically released when the bond is broken. More generally, you may need to thermostat your system to compensate for energy changes resulting from broken bonds.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix.

This fix computes two statistics which it stores in a global vector of length 2, which can be accessed by various output commands. The vector values calculated by this fix are "intensive".

These are the 2 quantities:

- (1) # of bonds broken on the most recent breakage timestep
- (2) cummulative # of bonds broken

No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions:

This fix is part of the "mc" package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

Currently, there are 2 restrictions for using this fix. We may relax these in the future if there are new models that would be enabled by it.

When a bond is broken, you might wish to turn off angle and dihedral interactions that include that bond. However, LAMMPS does not check for these angles and dihedrals, even if your simulation defines an angle_style or dihedral_style.

This fix requires that the pairwise weightings defined by the special_bonds command be 0,1,1 for 1−2, 1−3, and 1−4 neighbors within the bond topology. This effectively means that the pairwise interaction between atoms I and J is turned off when a bond between them exists and will be turned on when the bond is broken. It also means that the pairwise interaction of I with J's other bond partners is unaffected by the existence of the bond.

Related commands:

fix bond/create, fix bond/swap, dump local, special_bonds

Default:

The option defaults are prob = 1.0.

fix bond/create command

Syntax:

```
fix ID group-ID bond/create Nevery itype jtype Rmin bondtype keyword values ...
```

- ID, group−ID are documented in fix command
- bond/create = style name of this fix command
- Nevery = attempt bond creation every this many steps
- itype,jtype = atoms of itype can bond to atoms of jtype
- Rmin = 2 atoms separated by less than Rmin can bond (distance units)
- bondtype = type of created bonds
- zero or more keyword/value pairs may be appended to args
- keyword = *iparam* or *jparam* or *prob*

```
  iparam values = maxbond, newtype
      maxbond = max # of bonds of bondtype the itype atom can have
      newtype = change the itype atom to this type when maxbonds exist
    jparam values = maxbond, newtype
      maxbond = max # of bonds of bondtype the jtype atom can have
      newtype = change the jtype atom to this type when maxbonds exist
    prob values = fraction seed
      fraction = create a bond with this probability if otherwise eligible
      seed = random number seed (positive integer)
```

Examples:

```
fix 5 all bond/create 10 1 2 0.8 1
fix 5 all bond/create 1 3 3 0.8 1 prob 0.5 85784 iparam 2 3
```

Description:

Create bonds between pairs of atoms as a simulation runs according to specified criteria. This can be used to model cross−linking of polymers, the formation of a percolation network, etc. In this context, a bond means an interaction between a pair of atoms computed by the bond_style command. Once the bond is created it will be permanently in place. This is different than a pairwise bond−order potential such as Tersoff or AIREBO which infers bonds and many−body interactions based on the current geometry of a small cluster of atoms and effectively creates and destroys bonds from timestep to timestep as atoms move.

A check for possible new bonds is performed every *Nevery* timesteps. If two atoms I,J are within a distance *Rmin* of each other, if I is of atom type *itype*, if J is of atom type *jtype*, if both I and J are in the specified fix group, if a bond does not already exist between I and J, and if both I and J meet their respective *maxbond* requirement (explained below), then I,J is labeled as a "possible" bond pair.

If several atoms are close to an atom, it may have multiple possible bond partners. Every atom checks its list of possible bond partners and labels the closest such partner as its "sole" bond partner. After this is done, if atom I has atom J as its sole partner, and atom J has atom I as its sole partner, then the I,J bond is "eligible" to be formed.

Note that these rules mean an atom will only be part of at most one created bond on a given timestep. It also means that if atom I chooses atom J as its sole partner, but atom J chooses atom K is its sole partner (due to Rjk < Rij), then this means atom I will not form a bond on this timestep, even if it has other possible bond partners.

It is permissible to have *itype* = *jtype*. *Rmin* must be <= the pairwise cutoff distance between *itype* and *jtype* atoms, as defined by the pair_style command.

The *iparam* and *jparam* keywords can be used to limit the bonding functionality of the participating atoms. Each atom keeps track of how many bonds of *bondtype* it already has. If atom I of itype already has *maxbond* bonds (as set by the *iparam* keyword), then it will not form any more. Likewise for atom J. If *maxbond* is set to 0, then there is no limit on the number of bonds that can be formed with that atom.

The *newtype* value for *iparam* and *jparam* can be used to change the atom type of atom I or J when it reaches *maxbond* number of bonds of type *bondtype*. This means it can now interact in a pairwise fashion with other atoms in a different way by specifying different pair_coeff coefficients. If you do not wish the atom type to change, simply specify *newtype* as *itype* or *jtype*.

The *prob* keyword can also effect whether an eligible bond is actually created. The *fraction* setting must be a value between 0.0 and 1.0. A uniform random number between 0.0 and 1.0 is generated and the eligible bond is only created if the random number < fraction.

Any bond that is created is assigned a bond type of *bondtype*. Data structures within LAMMPS that store bond topology are updated to reflect the new bond. This can also affect subsequent computation of pairwise interactions involving the atoms in the bond. See the Restriction section below for additional information.

IMPORTANT NOTE: To create a new bond, the internal LAMMPS data structures that store this information must have space for it. When LAMMPS is initialized from a data file, the list of bonds is scanned and the maximum number of bonds per atom is tallied. If some atom will acquire more bonds than this limit as this fix operates, then the "extra bonds per atom" parameter in the data file header must be set to allow for it. See the read_data command for more details. Note that if this parameter needs to be set, it means a data file must be used to initialize the system, even if it initially has no bonds. A data file with no atoms can be used if you wish to add unbonded atoms via the create atoms command, e.g. for a percolation simulation.

IMPORTANT NOTE: LAMMPS also maintains a data structure that stores a list of 1st, 2nd, and 3rd neighbors of each atom (in the bond topology of the system) for use in weighting pairwise interactions for bonded atoms. Adding a bond adds a single entry to this list. The "extra" keyword of the special_bonds command should be used to leave space for new bonds if the maximum number of entries for any atom will be exceeded as this fix operates. See the special_bonds command for details.

Note that even if your simulation starts with no bonds, you must define a bond_style and use the bond_coeff command to specify coefficients for the *bondtype*. Similarly, if new atom types are specified by the *iparam* or *jparam* keywords, they must be within the range of atom types allowed by the simulation and pairwise coefficients must be specified for the new types.

Computationally, each timestep this fix operates, it loops over neighbor lists and computes distances between pairs of atoms in the list. It also communicates between neighboring processors to coordinate which bonds are created. Thus it roughly doubles the cost of a timestep. Thus you should be cautious about invoking this fix too frequently.

You can dump out snapshots of the current bond topology via the dump local command.

IMPORTANT NOTE: Creating a bond typically alters the energy of a system. You should be careful not to choose bond creation criteria that induce a dramatic change in energy. For example, if you define a very stiff harmonic bond and create it when 2 atoms are separated by a distance far from the equilibribum bond length, then the 2 atoms will oscillate dramatically when the bond is formed. More generally, you may need to thermostat your system to compensate for energy changes resulting from created bonds.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix.

This fix computes two statistics which it stores in a global vector of length 2, which can be accessed by various output commands. The vector values calculated by this fix are "intensive".

These are the 2 quantities:

- (1) # of bonds created on the most recent creation timestep
- (2) cummulative # of bonds created

No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions:

This fix is part of the "mc" package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

Currently, there are 2 restrictions for using this fix. We may relax these in the future if there are new models that would be enabled by it.

When a bond is created, you might wish to induce new angle and dihedral interactions that include that bond. However, LAMMPS does not create these angles and dihedrals, even if your simulation defines an angle_style or dihedral_style.

This fix requires that the pairwise weightings defined by the special_bonds command be 0,1,1 for 1−2, 1−3, and 1−4 neighbors within the bond topology. This effectively means that the pairwise interaction between atoms I and J will be turned off when a bond between them is created. It also means that the pairwise interaction of I with J's other bond partners will be unaffected by the new bond.

Related commands:

fix bond/break, fix bond/swap, dump local, special_bonds

Default:

The option defaults are iparam = (0,itype), jparam = (0,jtype), and prob = 1.0.

fix bond/swap command

Syntax:

```
fix ID group-ID bond/swap fraction cutoff seed
```

- ID, group−ID are documented in fix command
- bond/swap = style name of this fix command
- fraction = fraction of group atoms to consider for swapping
- cutoff = distance at which swapping will be considered (distance units)
- seed = random # seed (positive integer)

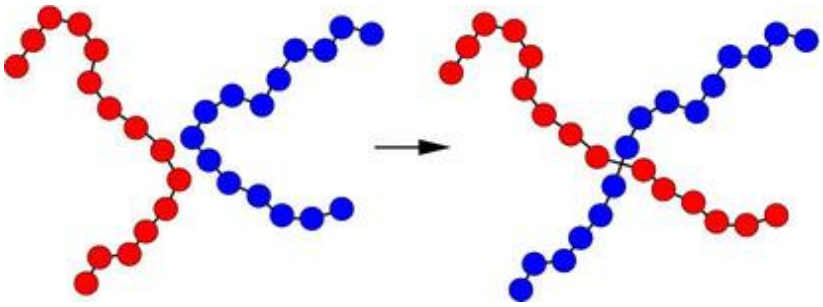
Examples:

```
fix 1 all bond/swap 0.5 1.3 598934
```

Description:

In a simulation of polymer chains, this command attempts to swap bonds between two different chains, effectively grafting the end of one chain onto another chain and vice versa. This is done via Monte Carlo rules using the Boltzmann acceptance criterion. The purpose is to equilibrate the polymer chain conformations more rapidly than dynamics alone would do it, by enabling instantaneous large conformational changes in a dense polymer melt. The polymer chains should thus more rapidly converge to the proper end−to−end distances and radii of gyration. It is designed for use with systems of FENE or harmonic bead−spring polymer chains where each polymer is a linear chain of monomers, but LAMMPS does not enforce this requirement, i.e. any bond_style can be used.

A schematic of the kinds of bond swaps that can occur is shown here:



On the left, the red and blue chains have two monomers A1 and B1 close to each other, which are currently bonded to monomers A2 and B2 respectively within their own chains. The bond swap operation will attempt to delete the A1−A2 and B1−B2 bonds and replace them with A1−B2 and B1−A2 bonds. If the swap is energetically favorable, the two chains on the right are the result and each polymer chain has undergone a dramatic conformational change. This reference provides more details on how the algorithm works and its application: (Sides).

The bond swapping operation is invoked each time neighbor lists are built during a simulation, since it potentially alters the list of which neighbors are considered for pairwise interaction. At each reneighboring step, each processor considers a random specified *fraction* of its atoms as potential swapping monomers for this timestep. Choosing a small *fraction* value can reduce the likelihood of a reverse swap occurring soon after an initial swap.

For each monomer A1, its neighbors are examined to find a possible B1 monomer. Both A1 and B1 must be in the fix group, their separation must be less than the specified *cutoff*, and the molecule IDs of A1 and B1 must be the same (see below). If a suitable partner is found, the energy change due to swapping the 2 bonds is computed. This includes changes in pairwise, bond, and angle energies due to the altered connectivity of the 2 chains. Dihedral and improper interactions are not allowed to be defined when this fix is used.

If the energy decreases due to the swap operation, the bond swap is accepted. If the energy increases it is accepted with probability exp(−delta/kT) where delta is the increase in energy, k is the Boltzmann constant, and T is the current temperature of the system. Whether the swap is accepted or rejected, no other swaps are attempted by this processor on this timestep.

The criterion for matching molecule IDs is how bond swaps performed by this fix conserve chain length. To use this features you must setup the molecule IDs for your polymer chains in a certain way, typically in the data file, read by the read_data comand. Consider a system of 6−mer chains. You have 3 choices. If the molecule IDs for monomers on each chain are set to 1,2,3,4,5,6 then swaps will conserve length. For a particular momoner there will be only one other monomer on another chain which is a potential swap partner. If the molecule IDs for monomers on each chain are set to 1,2,3,3,2,1 then swaps will conserve length but swaps will be able to occur at either end of a chain. Thus for a particular monomer there will be 2 possible swap partners on another chain. In this scenario, swaps can also occur within a single chain, i.e. the two ends of a chain swap with each other. The third choice is to give all monomers on all chains the same molecule ID, e.g. 0. This will allow a wide variety of swaps to occur, but will NOT conserve chain lengths.

IMPORTANT NOTE: If your simulation uses molecule IDs in the usual way, where all monomers on a single chain are assigned the same ID (different for each chain), then swaps will only occur within the same chain and will NOT conserve chain length. This is probably not what you want for this fix.

This fix computes a temperature each time it is invoked for use by the Boltzmann criterion. To do this, the fix creates its own compute of style *temp*, as if this command had been issued:

```
compute fix−ID_temp all temp
```

See the compute temp command for details. Note that the ID of the new compute is the fix−ID with underscore + "temp" appended and the group for the new compute is "all", so that the temperature of the entire system is used.

Note that this is NOT the compute used by thermodynamic output (see the thermo_style command) with ID = *thermo_temp*. This means you can change the attributes of this fix's temperature (e.g. its degrees−of−freedom) via the compute_modify command or print this temperature during thermodyanmic output via the thermo_style custom command using the appropriate compute−ID. It also means that changing attributes of *thermo_temp* will have no effect on this fix.

Restart, fix_modify, thermo output, run start/stop, minimize info:

No information about this fix is written to binary restart files. Because the state of the random number generator is not saved in restart files, this means you cannot do "exact" restarts with this fix, where the simulation continues on the same as if no restart had taken place. However, in a statistical sense, a restarted simulation should produce the same behavior. Also note that each processor generates possible swaps independently of other processors. Thus if you repeat the same simulation on a different number of processors, the specific swaps performed will be different.

The fix_modify *temp* option is supported by this fix. You can use it to assign a compute you have defined to this fix which will be used to compute the temperature for the Boltzmann criterion.

This fix computes two statistical quantities as a global 2−vector of output, which can be accessed by various output commands. The first component of the vector is the cummulative number of swaps performed by all processors. The second component of the vector is the cummulative number of swaps attempted (whether accepted or rejected). Note that a swap "attempt" only occurs when swap partners meeting the criteria described above are found on a particular timestep. The vector values calculated by this fix are "intensive".

No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions:

This fix is part of the "mc" package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

The setings of the "special_bond" command must be 0,1,1 in order to use this fix, which is typical of bead−spring chains with FENE or harmonic bonds. This means that pairwise interactions between bonded atoms are turned off, but are turned on between atoms two or three hops away along the chain backbone.

Currently, energy changes in dihedral and improper interactions due to a bond swap are not considered. Thus a simulation that uses this fix cannot use a dihedral or improper potential.

Related commands: none

Default: none

(Sides) Sides, Grest, Stevens, Plimpton, J Polymer Science B, 42, 199−208 (2004).

fix box/relax command

Syntax:

```
fix ID group-ID box/relax keyword value ...
```

- ID, group−ID are documented in fix command
- box/relax = style name of this fix command

```
one or more keyword value pairs may be appended
keyword = iso or aniso or tri or x or y or z or xy or yz or xz or couple or nreset or vmax or
  iso or aniso or tri value = Ptarget = desired pressure (pressure units)
  x or y or z or xy or yz or xz value = Ptarget = desired pressure (pressure units)
  couple = none or xyz or xy or yz or xz
  nreset value = reset reference cell every this many minimizer iterations
  vmax value = fraction = max allowed volume change in one iteration
  dilate value = all or partial
```

Examples:

```
fix 1 all box/relax iso 0.0 vmax 0.001
fix 2 water box/relax aniso 0.0 dilate partial
fix 2 ice box/relax tri 0.0 couple xy nreset 100
```

Description:

Apply an external pressure or stress tensor to the simulation box during an energy minimization. This allows the box size and shape to vary during the iterations of the minimizer so that the final configuration will be both an energy minimum for the potential energy of the atoms, and the system pressure tensor will be close to the specified external tensor. Conceptually, specifying a positive pressure is like squeezing on the simulation box; a negative pressure typically allows the box to expand.

The external pressure tensor is specified using one or more of the *iso*, *aniso*, *tri*, *x*, *y*, *z*, *xy*, *xz*, *yz*, and *couple* keywords. These keywords give you the ability to specify all 6 components of an external stress tensor, and to couple various of these components together so that the dimensions they represent are varied together during the mimimization.

Orthogonal simulation boxes have 3 adjustable dimensions (x,y,z). Triclinic (non−orthogonal) simulation boxes have 6 adjustable dimensions (x,y,z,xy,xz,yz). The create_box, read data, and read_restart commands specify whether the simulation box is orthogonal or non−orthogonal (triclinic) and explain the meaning of the xy,xz,yz tilt factors.

The target pressures *Ptarget* for each of the 6 components of the stress tensor can be specified independently via the *x*, *y*, *z*, *xy*, *xz*, *yz* keywords, which correspond to the 6 simulation box dimensions. For example, if the *y* keyword is used, the y−box length will change during the minimization. If the *xy* keyword is used, the xy tilt factor will change. A box dimension will not change if that component is not specified.

Note that in order to use the *xy*, *xz*, or *yz* keywords, the simulation box must be triclinic, even if its initial tilt factors are 0.0.

When the size of the simulation box changes, all atoms are re−scaled to new positions, unless the keyword *dilate* is specified with a value of *partial*, in which case only the atoms in the fix group are re−scaled. This can be useful

for leaving the coordinates of atoms in a solid substrate unchanged and controlling the pressure of a surrounding fluid.

IMPORTANT NOTE: Appling an external pressure to tilt dimensions *xy*, *xz*, *yz* can sometimes result in arbitrarily large values of the tilt dimensions, i.e. a dramatically deformed simulation box. This typically indicates that there is something badly wrong with how the simulation was constructed. The two most common sources of this error are applying a shear stress to a liquid system or specifying an external shear stress tensor that exceeds the yield stress of the solid. In either case the minimization is either not going to converge at all, or converge to a bogus conformation. Note that LAMMPS will not throw an error when the tilt value becomes extreme, but the final box may be unsuitable for running dynamics on, unless fix deform is used first to remap the box to a valid tilt value.

The *couple* keyword allows two or three of the diagonal components of the pressure tensor to be "coupled" together. The value specified with the keyword determines which are coupled. For example, *xz* means the *Pxx* and *Pzz* components of the stress tensor are coupled. *Xyz* means all 3 diagonal components are coupled. Coupling means two things: the instantaneous stress will be computed as an average of the corresponding diagonal components, and the coupled box dimensions will be changed together in lockstep, meaning coupled dimensions will be dilated or contracted by the same percentage every timestep. The *Ptarget* values for any coupled dimensions must be identical. *Couple xyz* can be used for a 2d simulation; the *z* dimension is simply ignored.

The *iso*, *aniso*, and *tri* keywords are simply shortcuts that are equivalent to specifying several other keywords together.

The keyword *iso* means couple all 3 diagonal components together when pressure is computed (hydrostatic pressure), and dilate/contract the dimensions together. Using "iso Ptarget" is the same as specifying these 4 keywords:

```
x Ptarget
y Ptarget
z Ptarget
couple xyz
```

The keyword *aniso* means *x*, *y*, and *z* dimensions are controlled independently using the *Pxx*, *Pyy*, and *Pzz* components of the stress tensor as the driving forces, and the specified scalar external pressure. Using "aniso Ptarget" is the same as specifying these 4 keywords:

```
x Ptarget
y Ptarget
z Ptarget
couple none
```

The keyword *tri* means *x*, *y*, *z*, *xy*, *xz*, and *yz* dimensions are controlled independently using their individual stress components as the driving forces, and the specified scalar pressure as the external normal stress. Using "tri Ptarget" is the same as specifying these 7 keywords:

```
x Ptarget
y Ptarget
z Ptarget
xy 0.0
yz 0.0
xz 0.0
couple none
```

The *vmax* keyword can be used to limit the fractional change in the volume of the simulation box that can occur in one iteration of the minimizer. If the pressure is not settling down during the minimization this can be because the volume is fluctuating too much. The specified fraction must be greater than 0.0 and should be << 1.0. A value of

0.001 means the volume cannot change by more than 1/10 of a percent in one iteration when *couple xyz* has been specified. For any other case it means no linear dimension of the simulation box can change by more than 1/10 of a percent.

With this fix, the potential energy used by the minimizer is augmented by an additional energy provided by the fix. The overall objective function then is:

$$E = U + P_t \left(V - V_0\right) + E_{strain}$$

where U is the system potential energy, P_t is the desired hydrostatic pressure, V and V_0 are the system and reference volumes, respectively. *E_strain* is the strain energy expression proposed by Parrinello and Rahman (Parrinello1981). Taking derivatives of E w.r.t. the box dimensions, and setting these to zero, we find that at the minimum of the objective function, the global system stress tensor **P** will satisfy the relation:

$$\mathbf{P} = P_t \mathbf{I} + \mathbf{S}_t \left(\mathbf{h}_0^{-1}\right)^t \mathbf{h}_{0d}$$

where **I** is the identity matrix, **h**_0 is the box dimension tensor of the reference cell, and **h**_0*d* is the diagonal part of **h**_0. **S**_*t* is a symmetric stress tensor that is chosen by LAMMPS so that the upper−triangular components of **P** equal the stress tensor specified by the user.

This equation only applies when the box dimensions are equal to those of the reference dimensions. If this is not the case, then the converged stress tensor will not equal that specified by the user. We can resolve this problem by periodically resetting the reference dimensions. The keyword *nreset_ref* controls how often this is done. If this keyword is not used, or is given a value of zero, then the reference dimensions are set to those of the initial simulation domain and are never changed. A value of *nstep* means that every *nstep* minimization steps, the reference dimensions are set to those of the current simulation domain. Note that resetting the reference dimensions changes the objective function and gradients, which sometimes causes the minimization to fail. This can be resolved by changing the value of *nreset*, or simply continuing the minimization from a restart file.

IMPORTANT NOTE: As normally computed, pressure includes a kinetic− energy or temperature−dependent component; see the compute pressure command. However, atom velocities are ignored during a minimization, and the applied pressure(s) specified with this command are assumed to only be the virial component of the pressure (the non−kinetic portion). Thus if atoms have a non−zero temperature and you print the usual thermodynamic pressure, it may not appear the system is converging to your specified pressure. The solution for this is to either (a) zero the velocities of all atoms before performing the minimization, or (b) make sure you are monitoring the pressure without its kinetic component. The latter can be done by outputting the pressure from the fix this command creates (see below) or a pressure fix you define yourself.

IMPORTANT NOTE: Because pressure is often a very sensitive function of volume, it can be difficult for the minimizer to equilibrate the system the desired pressure with high precision, particularly for solids. Some techniques that seem to help are (a) use the "min_modify line quadratic" option when minimizing with box relaxations, and (b) minimize several times in succession if need be, to drive the pressure closer to the target pressure. Also note that some systems (e.g. liquids) will not sustain a non−hydrostatic applied pressure, which means the minimizer will not converge.

This fix computes a temperature and pressure each timestep. The temperature is used to compute the kinetic contribution to the pressure, even though this is subsequently ignored by default. To do this, the fix creates its own computes of style "temp" and "pressure", as if these commands had been issued:

```
compute fix−ID_temp group−ID temp
```

```
compute fix-ID_press group-ID pressure fix-ID_temp virial
```

See the compute temp and compute pressure commands for details. Note that the IDs of the new computes are the fix−ID + underscore + "temp" or fix_ID + underscore + "press", and the group for the new computes is the same as the fix group. Also note that the pressure compute does not include a kinetic component.

Note that these are NOT the computes used by thermodynamic output (see the thermo_style command) with ID = *thermo_temp* and *thermo_press*. This means you can change the attributes of this fix's temperature or pressure via the compute_modify command or print this temperature or pressure during thermodynamic output via the thermo_style custom command using the appropriate compute−ID. It also means that changing attributes of *thermo_temp* or *thermo_press* will have no effect on this fix.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files.

The fix_modify *temp* and *press* options are supported by this fix. You can use them to assign a compute you have defined to this fix which will be used in its temperature and pressure calculation, as described above. Note that as described above, if you assign a pressure compute to this fix that includes a kinetic energy component it will affect the minimization, most likely in an undesirable way.

IMPORTANT NOTE: If both the *temp* and *press* keywords are used in a single thermo_modify command (or in two separate commands), then the order in which the keywords are specified is important. Note that a pressure compute defines its own temperature compute as an argument when it is specified. The *temp* keyword will override this (for the pressure compute being used by fix npt), but only if the *temp* keyword comes after the *press* keyword. If the *temp* keyword comes before the *press* keyword, then the new pressure compute specified by the *press* keyword will be unaffected by the *temp* setting.

This fix computes a global scalar which can be accessed by various output commands. The scalar is the pressure−volume energy, plus the strain energy, if it exists.

No parameter of this fix can be used with the *start/stop* keywords of the run command.

This fix is invoked during energy minimization, but not for the purpose of adding a contribution to the energy or forces being minimized. Instead it alters the simulation box geometry as described above.

Restrictions:

Only dimensions that are available can be adjusted by this fix. Non−periodic dimensions are not available. *z*, *xz*, and *yz*, are not available for 2D simulations. *xy*, *xz*, and *yz* are only available if the simulation domain is non−orthogonal. The create_box, read data, and read_restart commands specify whether the simulation box is orthogonal or non−orthogonal (triclinic) and explain the meaning of the xy,xz,yz tilt factors.

Related commands:

fix npt, minimize

Default:

The keyword defaults are dilate = all, vmax = 0.0001, nreset = 0.

(Parrinello1981) Parrinello and Rahman, J Appl Phys, 52, 7182 (1981).

fix deform command

Syntax:

```
fix ID group-ID deform N parameter args ... keyword value ...
```

- ID, group−ID are documented in fix command
- deform = style name of this fix command
- N = perform box deformation every this many timesteps
- one or more parameter/arg pairs may be appended

```
parameter = x or y or z or xy or xz or yz
  x, y, z args = style value(s)
    style = final or delta or scale or vel or erate or trate or volume or wiggle
      final values = lo hi
        lo hi = box boundaries at end of run (distance units)
      delta values = dlo dhi
        dlo dhi = change in box boundaries at end of run (distance units)
      scale values = factor
        factor = multiplicative factor for change in box length at end of run
      vel value = V
        V = change box length at this velocity (distance/time units),
            effectively an engineering strain rate
      erate value = R
        R = engineering strain rate (1/time units)
      trate value = R
        R = true strain rate (1/time units)
      volume value = none = adjust this dim to preserve volume of system
      wiggle value = A Tp
        A = amplitude of oscillation (distance units)
        Tp = period of oscillation (time units)
  xy, xz, yz args = style value
    style = final or delta or vel or erate or trate or wiggle
      final value = tilt
        tilt = tilt factor at end of run (distance units)
      delta value = dtilt
        dtilt = change in tilt factor at end of run (distance units)
      vel value = V
        V = change tilt factor at this velocity (distance/time units),
            effectively an engineering shear strain rate
      erate value = R
        R = engineering shear strain rate (1/time units)
      trate value = R
        R = true shear strain rate (1/time units)
      wiggle value = A Tp
        A = amplitude of oscillation (distance units)
        Tp = period of oscillation (time units)
```

- zero or more keyword/value pairs may be appended
- keyword = *remap* or *units*

```
remap value = x or v or none
    x = remap coords of atoms in group into deforming box
    v = remap velocities of all atoms when they cross periodic boundaries
    none = no remapping of x or v
  units value = lattice or box
    lattice = distances are defined in lattice units
    box = distances are defined in simulation box units
```

Examples:

```
fix 1 all deform 1 x final 0.0 9.0 z final 0.0 5.0 units box
fix 1 all deform 1 x trate 0.1 y volume z volume
fix 1 all deform 1 xy erate 0.001 remap v
fix 1 all deform 10 y delta −0.5 0.5 xz vel 1.0
```

Description:

Change the volume and/or shape of the simulation box during a dynamics run. Orthogonal simulation boxes have 3 adjustable parameters (x,y,z). Triclinic (non−orthogonal) simulation boxes have 6 adjustable parameters (x,y,z,xy,xz,yz). Any or all of them can be adjusted independently and simultaneously by this command. This fix can be used to perform non−equilibrium MD (NEMD) simulations of a continuously strained system. See the fix nvt/sllod and compute temp/deform commands for more details.

Any parameter varied by this command must refer to a periodic dimension − see the boundary command. For parameters *xy*, *xz*, and *yz*, the 2nd dimension must be periodic, e.g. *y* for *xy*. Dimensions not varied by this command can be periodic or non−periodic. Dimensions corresponding to unspecified parameters can also be controlled by a fix npt or fix nph command.

The size and shape of the simulation box at the beginning of the simulation run were either specified by the create_box or read_data or read_restart command used to setup the simulation initially if it is the first run, or they are the values from the end of the previous run. The create_box, read data, and read_restart commands specify whether the simulation box is orthogonal or non−orthogonal (triclinic) and explain the meaning of the xy,xz,yz tilt factors. If fix deform changes the xy,xz,yz tilt factors, then the simulation box must be triclinic, even if its initial tilt factors are 0.0.

As described below, the desired simulation box size and shape at the end of the run are determined by the parameters of the fix deform command. Every Nth timestep during the run, the simulation box is expanded, contracted, or tilted to ramped values between the initial and final values.

For the *x*, *y*, and *z* parameters, this is the meaning of their styles and values.

The *final*, *delta*, *scale*, *vel*, and *erate* styles all change the specified dimension of the box via "constant displacement" which is effectively a "constant engineering strain rate". This means the box dimension changes linearly with time from its initial to final value.

For style *final*, the final lo and hi box boundaries of a dimension are specified. The values can be in lattice or box distance units. See the discussion of the units keyword below.

For style *delta*, plus or minus changes in the lo/hi box boundaries of a dimension are specified. The values can be in lattice or box distance units. See the discussion of the units keyword below.

For style *scale*, a multiplicative factor to apply to the box length of a dimension is specified. For example, if the initial box length is 10, and the factor is 1.1, then the final box length will be 11. A factor less than 1.0 means compression.

For style *vel*, a velocity at which the box length changes is specified in units of distance/time. This is effectively a "constant engineering strain rate", where rate = V/L0 and L0 is the initial box length. The distance can be in lattice or box distance units. See the discussion of the units keyword below. For example, if the initial box length is 100 Angstroms, and V is 10 Angstroms/psec, then after 10 psec, the box length will have doubled. After 20 psec, it will have tripled.

The *erate* style changes a dimension of the the box at a "constant engineering strain rate". The units of the specified strain rate are 1/time. See the units command for the time units associated with different choices of simulation units, e.g. picoseconds for "metal" units). Tensile strain is unitless and is defined as delta/L0, where L0 is the original box length and delta is the change relative to the original length. The box length L as a function of time will change as

```
L(t) = L0 (1 + erate*dt)
```

where dt is the elapsed time (in time units). Thus if *erate* R is specified as 0.1 and time units are picoseconds, this means the box length will increase by 10% of its original length every picosecond. I.e. strain after 1 psec = 0.1, strain after 2 psec = 0.2, etc. R = −0.01 means the box length will shrink by 1% of its original length every picosecond. Note that for an "engineering" rate the change is based on the original box length, so running with R = 1 for 10 picoseconds expands the box length by a factor of 11 (strain of 10), which is different that what the *trate* style would induce.

The *trate* style changes a dimension of the box at a "constant true strain rate". Note that this is not an "engineering strain rate", as the other styles are. Rather, for a "true" rate, the rate of change is constant, which means the box dimension changes non−linearly with time from its initial to final value. The units of the specified strain rate are 1/time. See the units command for the time units associated with different choices of simulation units, e.g. picoseconds for "metal" units). Tensile strain is unitless and is defined as delta/L0, where L0 is the original box length and delta is the change relative to the original length.

The box length L as a function of time will change as

```
L(t) = L0 exp(trate*dt)
```

where dt is the elapsed time (in time units). Thus if *trate* R is specified as ln(1.1) and time units are picoseconds, this means the box length will increase by 10% of its current (not original) length every picosecond. I.e. strain after 1 psec = 0.1, strain after 2 psec = 0.21, etc. R = ln(2) or ln(3) means the box length will double or triple every picosecond. R = ln(0.99) means the box length will shrink by 1% of its current length every picosecond. Note that for a "true" rate the change is continuous and based on the current length, so running with R = ln(2) for 10 picoseconds does not expand the box length by a factor of 11 as it would with *erate*, but by a factor of 1024 since the box length will double every picosecond.

Note that to change the volume (or cross−sectional area) of the simulation box at a constant rate, you can change multiple dimensions via *erate* or *trate*. E.g. to double the box volume in a picosecond picosecond, you could set "x erate M", "y erate M", "z erate M", with M = pow(2,1/3) − 1 = 0.26, since if each box dimension grows by 26%, the box volume doubles. Or you could set "x trate M", "y trate M", "z trate M", with M = ln(1.26) = 0.231, and the box volume would double every picosecond.

The *volume* style changes the specified dimension in such a way that the box volume remains constant while other box dimensions are changed explicitly via the styles discussed above. For example, "x scale 1.1 y scale 1.1 z volume" will shrink the z box length as the x,y box lengths increase, to keep the volume constant (product of x,y,z lengths). If "x scale 1.1 z volume" is specified and parameter *y* is unspecified, then the z box length will shrink as x increases to keep the product of x,z lengths constant. If "x scale 1.1 y volume z volume" is specified, then both the y,z box lengths will shrink as x increases to keep the volume constant (product of x,y,z lengths). In this case, the y,z box lengths shrink so as to keep their relative aspect ratio constant.

For solids or liquids, note that when one dimension of the box is expanded via fix deform (i.e. tensile strain), it may be physically undesirable to hold the other 2 box lengths constant (unspecified by fix deform) since that implies a density change. Using the *volume* style for those 2 dimensions to keep the box volume constant may make more physical sense, but may also not be correct for materials and potentials whose Poisson ratio is not 0.5. An alternative is to use fix npt aniso with zero applied pressure on those 2 dimensions, so that they respond to the

tensile strain dynamically.

The *wiggle* style oscillates the specified box length dimension sinusoidally with the specified amplitude and period. I.e. the box length L as a function of time is given by

```
L(t) = L0 + A sin(2*pi t/Tp)
```

where L0 is its initial length. If the amplitude A is a positive number the box initially expands, then contracts, etc. If A is negative then the box initially contracts, then expands, etc. The amplitude can be in lattice or box distance units. See the discussion of the units keyword below.

For the *scale*, *vel*, *erate*, *trate*, *volume*, and *wiggle* styles, the box length is expanded or compressed around its mid point.

For the *xy*, *xz*, and *yz* parameters, this is the meaning of their styles and values. Note that changing the tilt factors of a triclinic box does not change its volume.

The *final*, *delta*, *vel*, and *erate* styles all change the shear strain at a "constant engineering shear strain rate". This means the tilt factor changes linearly with time from its initial to final value.

For style *final*, the final tilt factor is specified. The value can be in lattice or box distance units. See the discussion of the units keyword below.

For style *delta*, a plus or minus change in the tilt factor is specified. The value can be in lattice or box distance units. See the discussion of the units keyword below.

For style *vel*, a velocity at which the tilt factor changes is specified in units of distance/time. This is effectively an "engineering shear strain rate", where rate = V/L0 and L0 is the initial box length perpendicular to the direction of shear. The distance can be in lattice or box distance units. See the discussion of the units keyword below. For example, if the initial tilt factor is 5 Angstroms, and the V is 10 Angstroms/psec, then after 1 psec, the tilt factor will be 15 Angstroms. After 2 psec, it will be 25 Angstroms.

The *erate* style changes a tilt factor at a "constant engineering shear strain rate". The units of the specified shear strain rate are 1/time. See the units command for the time units associated with different choices of simulation units, e.g. picoseconds for "metal" units). Shear strain is unitless and is defined as offset/length, where length is the box length perpendicular to the shear direction (e.g. y box length for xy deformation) and offset is the displacement distance in the shear direction (e.g. x direction for xy deformation) from the unstrained orientation.

The tilt factor T as a function of time will change as

```
T(t) = T0 + erate*dt
```

where T0 is the initial tilt factor and dt is the elapsed time (in time units). Thus if *erate* R is specified as 0.1 and time units are picoseconds, this means the shear strain will increase by 0.1 every picosecond. I.e. if the xy shear strain was initially 0.0, then strain after 1 psec = 0.1, strain after 2 psec = 0.2, etc. Thus the tilt factor would be 0.0 at time 0, 0.1*ybox at 1 psec, 0.2*ybox at 2 psec, etc, where ybox is the original y box length. R = 1 or 2 means the tilt factor will increase by 1 or 2 every picosecond. R = −0.01 means a decrease in shear strain by 0.01 every picosecond.

The *trate* style changes a tilt factor at a "constant true shear strain rate". Note that this is not an "engineering shear strain rate", as the other styles are. Rather, for a "true" rate, the rate of change is constant, which means the tilt factor changes non−linearly with time from its initial to final value. The units of the specified shear strain rate are 1/time. See the units command for the time units associated with different choices of simulation units, e.g.

picoseconds for "metal" units). Shear strain is unitless and is defined as offset/length, where length is the box length perpendicular to the shear direction (e.g. y box length for xy deformation) and offset is the displacement distance in the shear direction (e.g. x direction for xy deformation) from the unstrained orientation.

The tilt factor T as a function of time will change as

```
T(t) = T0 exp(trate*dt)
```

where T0 is the initial tilt factor and dt is the elapsed time (in time units). Thus if *trate* R is specified as ln(1.1) and time units are picoseconds, this means the shear strain or tilt factor will increase by 10% every picosecond. I.e. if the xy shear strain was initially 0.1, then strain after 1 psec = 0.11, strain after 2 psec = 0.121, etc. R = ln(2) or ln(3) means the tilt factor will double or triple every picosecond. R = ln(0.99) means the tilt factor will shrink by 1% every picosecond. Note that the change is continuous, so running with R = ln(2) for 10 picoseconds does not change the tilt factor by a factor of 10, but by a factor of 1024 since it doubles every picosecond. Note that the initial tilt factor must be non−zero to use the *trate* option.

Note that shear strain is defined as the tilt factor divided by the perpendicular box length. The *erate* and *trate* styles control the tilt factor, but assume the perpendicular box length remains constant. If this is not the case (e.g. it changes due to another fix deform parameter), then this effect on the shear strain is ignored.

The *wiggle* style oscillates the specified tilt factor sinusoidally with the specified amplitude and period. I.e. the tilt factor T as a function of time is given by

```
T(t) = T0 + A sin(2*pi t/Tp)
```

where T0 is its initial value. If the amplitude A is a positive number the tilt factor initially becomes more positive, then more negative, etc. If A is negative then the tilt factor initially becomes more negative, then more positive, etc. The amplitude can be in lattice or box distance units. See the discussion of the units keyword below.

All of these styles change the xy, xz, yz tilt factors during a simulation. In LAMMPS, tilt factors (xy,xz,yz) for triclinic boxes are always bounded by half the distance of the parallel box length. For example, if xlo = 2 and xhi = 12, then the x box length is 10 and the xy tilt factor must be between −5 and 5. Similarly, both xz and yz must be between −(xhi−xlo)/2 and +(yhi−ylo)/2. Note that this is not a limitation, since if the maximum tilt factor is 5 (as in this example), then configurations with tilt = ..., −15, −5, 5, 15, 25, ... are all equivalent.

To obey this constraint and allow for large shear deformations to be applied via the *xy*, *xz*, or *yz* parameters, the following algorithm is used. If *prd* is the associated parallel box length (10 in the example above), then if the tilt factor exceeds the accepted range of −5 to 5 during the simulation, then the box is re−shaped to the other limit (an equivalent box) and the simulation continues. Thus for this example, if the initial xy tilt factor was 0.0 and "xy final 100.0" was specified, then during the simulation the xy tilt factor would increase from 0.0 to 5.0, the box would be re−shaped so that the tilt factor becomes −5.0, the tilt factor would increase from −5.0 to 5.0, the box would be re−shaped again, etc. The re−shaping would occur 10 times and the final tilt factor at the end of the simulation would be 0.0. During each re−shaping event, atoms are remapped into the new box in the appropriate manner.

Each time the box size or shape is changed, the *remap* keyword determines whether atom positions are remapped to the new box. If *remap* is set to *x* (the default), atoms in the fix group are remapped; otherwise they are not. Note that their velocities are not changed, just their positions are altered. If *remap* is set to *v*, then any atom in the fix group that crosses a periodic boundary will have a delta added to its velocity equal to the difference in velocities between the lo and hi boundaries. Note that this velocity difference can include tilt components, e.g. a delta in the x velocity when an atom crosses the y periodic boundary. If *remap* is set to *none*, then neither of these remappings take place.

Conceptually, setting *remap* to *x* forces the atoms to deform via an affine transformation that exactly matches the box deformation. This setting is typically appropriate for solids. Note that though the atoms are effectively "moving" with the box over time, it is not due to their having a velocity that tracks the box change, but only due to the remapping. By contrast, setting *remap* to *v* is typically appropriate for fluids, where you want the atoms to respond to the change in box size/shape on their own and acquire a velocity that matches the box change, so that their motion will naturally track the box without explicit remapping of their coordinates.

IMPORTANT NOTE: When non−equilibrium MD (NEMD) simulations are performed using this fix, the option "remap v" should normally be used. This is because fix nvt/sllod adjusts the atom positions and velocities to induce a velocity profile that matches the changing box size/shape. Thus atom coordinates should NOT be remapped by fix deform, but velocities SHOULD be when atoms cross periodic boundaries, since that is consistent with maintaining the velocity profile already created by fix nvt/sllod. LAMMPS will warn you if the *remap* setting is not consistent with fix nvt/sllod.

IMPORTANT NOTE: If a fix rigid is defined for rigid bodies, and *remap* is set to *x*, then the center−of−mass coordinates of rigid bodies will be remapped to the changing simulation box. This will be done regardless of whether atoms in the rigid bodies are in the fix deform group or not. The velocity of the centers of mass are not remapped even if *remap* is set to *v*, since fix nvt/sllod does not currently do anything special for rigid particles. If you wish to perform a NEMD simulation of rigid particles, you can either thermostat them independently or include a background fluid and thermostat the fluid via fix nvt/sllod.

The *units* keyword determines the meaning of the distance units used to define various arguments. A *box* value selects standard distance units as defined by the units command, e.g. Angstroms for units = real or metal. A *lattice* value means the distance units are in lattice spacings. The lattice command must have been previously used to define the lattice spacing. Note that the units choice also affects the *vel* style parameters since it is defined in terms of distance/time.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix. No global or per−atom quantities are stored by this fix for access by various output commands.

This fix can perform deformation over multiple runs, using the *start* and *stop* keywords of the run command. See the run command for details of how to do this.

This fix is not invoked during energy minimization.

Restrictions:

Any box dimension varied by this fix must be periodic.

Related commands:

displace_box

Default:

The option defaults are remap = x and units = lattice.

fix deposit command

Syntax:

```
fix ID group-ID deposit N type M seed keyword values ...
```

- ID, group−ID are documented in fix command
- deposit = style name of this fix command
- N = # of atoms to insert
- type = atom type to assign to inserted atoms
- M = insert a single particle every M steps
- seed = random # seed (positive integer)
- one or more keyword/value pairs may be appended to args
- keyword = *region* or *global* or *local* or *near* or *attempt* or *rate* or *vx* or *vy* or *vz* or *units*

```
  region value = region-ID
      region-ID = ID of region to use as insertion volume
    global values = lo hi
      lo,hi = put new particle a distance lo-hi above all other particles (distance units)
    local values = lo hi delta
      lo,hi = put new particle a distance lo-hi above any nearby particle beneath it (distance u
      delta = lateral distance within which a neighbor is considered "nearby" (distance units)
    near value = R
      R = only insert particle if further than R from existing particles (distance units)
    attempt value = Q
      Q = attempt a single insertion up to Q times
    rate value = V
      V = z velocity (y in 2d) at which insertion volume moves (velocity units)
    vx values = vxlo vxhi
      vxlo,vxhi = range of x velocities for inserted particle (velocity units)
    vy values = vylo vyhi
      vylo,vyhi = range of y velocities for inserted particle (velocity units)
    vz values = vzlo vzhi
      vzlo,vzhi = range of z velocities for inserted particle (velocity units)
    units value = lattice or box
      lattice = the geometry is defined in lattice units
      box = the geometry is defined in simulation box units
```

Examples:

```
fix 3 all deposit 1000 2 100 29494 region myblock local 1.0 1.0 1.0 units box
fix 2 newatoms deposit 10000 1 500 12345 region disk near 2.0 vz -1.0 -0.8
```

Description:

Insert a single particle into the simulation domain every M timesteps until N particles have been inserted. This is useful for simulating the deposition of particles onto a surface.

Inserted particles have the specified atom type and are assigned to two groups: the default group "all" and the group specified in the fix deposit command (which can also be "all").

If you are computing temperature values which include inserted particles, you will want to use the compute_modify dynamic option, which insures the current number of atoms is used as a normalizing factor each time temperature is computed.

Care must be taken that inserted particles are not too near existing particles, using the options described below. When inserting particles above a surface in a non−periodic box (see the boundary command), the possibility of a particle escaping the surface and flying upward should be considered, since the particle may be lost or the box size may grow infinitely large. A fix wall/reflect command can be used to prevent this behavior. Note that if a shrink−wrap boundary is used, it is OK to insert the new particle outside the box, however the box will immediately be expanded to include the new particle.

This command must use the *region* keyword to define an insertion volume. The specified region must have been previously defined with a region command. It must be defined with side = *in*.

Each timestep a particle is to be inserted, its coordinates are chosen as follows. A random position within the insertion volume is generated. If neither the *global* or *local* keyword is used, that is the trial position. If the *global* keyword is used, the random x,y values are used, but the z position of the new particle is set above the highest current atom in the simulation by a distance randomly chosen between lo/hi. (For a 2d simulation, this is done for the y position.) If the *local* keyword is used, the z position is set a distance between lo/hi above the highest current atom in the simulation that is "nearby" the chosen x,y position. In this context, "nearby" means the lateral distance (in x,y) between the new and old particles is less than the delta parameter.

Once a trial x,y,z location has been computed, the insertion is only performed if no current particle in the simulation is within a distance R of the new particle. If this test fails, a new random position within the insertion volume is chosen and another trial is made. Up to Q attempts are made. If an atom is not successfully deposited, LAMMPS prints a warning message.

The *rate* option moves the insertion volume in the z direction (3d) or y direction (2d). This enables particles to be inserted from a successively higher height over time. Note that this parameter is ignored if the *global* or *local* keywords are used, since those options choose a z−coordinate for insertion independently.

The vx, vy, and vz components of velocity for the inserted particle are set using the values specified for the *vx*, *vy*, and *vz* keywords. Note that normally, new particles should be a assigned a negative vertical velocity so that they move towards the surface.

The *units* keyword determines the meaning of the distance units used for the other deposition parameters. A *box* value selects standard distance units as defined by the units command, e.g. Angstroms for units = real or metal. A *lattice* value means the distance units are in lattice spacings. The lattice command must have been previously used to define the lattice spacing. Note that the units choice affects all the keyword values that have units of distance or velocity.

Restart, fix_modify, output, run start/stop, minimize info:

This fix writes the state of the deposition to binary restart files. This includes information about how many atoms have been depositied, the random number generator seed, the next timestep for deposition, etc. See the read_restart command for info on how to re−specify a fix in an input script that reads a restart file, so that the operation of the fix continues in an uninterrupted fashion.

None of the fix_modify options are relevant to this fix. No global or per−atom quantities are stored by this fix for access by various output commands. No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions:

The specified insertion region cannot be a "dynamic" region, as defined by the region command.

Related commands:

fix_pour, region

Default:

The option defaults are delta = 0.0, near = 0.0, attempt = 10, rate = 0.0, vx = 0.0 0.0, vy = 0.0 0.0, vz = 0.0 0.0, and units = lattice.

fix drag command

Syntax:

```
fix ID group−ID drag x y z fmag delta
```

- ID, group−ID are documented in fix command
- drag = style name of this fix command
- x,y,z = coord to drag atoms towards
- fmag = magnitude of force to apply to each atom (force units)
- delta = cutoff distance inside of which force is not applied (distance units)

Examples:

```
fix center small−molecule drag 0.0 10.0 0.0 5.0 2.0
```

Description:

Apply a force to each atom in a group to drag it towards the point (x,y,z). The magnitude of the force is specified by fmag. If an atom is closer than a distance delta to the point, then the force is not applied.

Any of the x,y,z values can be specified as NULL which means do not include that dimension in the distance calculation or force application.

This command can be used to steer one or more atoms to a new location in the simulation.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix.

This fix computes a global 3−vector of forces, which can be accessed by various output commands. This is the total force on the group of atoms by the drag force. The vector values calculated by this fix are "extensive".

No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions: none

Related commands:

fix spring, fix spring/self, fix spring/rg, fix smd

Default: none

fix dt/reset command

Syntax:

```
fix ID group−ID dt/reset N Tmin Tmax Xmax keyword values ...
```

- ID, group−ID are documented in fix command
- dt/reset = style name of this fix command
- N = recompute dt every N timesteps
- Tmin = minimum dt allowed (can be NULL) (time units)
- Tmax = maximum dt allowed (can be NULL) (time units)
- Xmax = maximum distance for an atom to move in one timestep (distance units)
- zero or more keyword/value pairs may be appended
- keyword = *units*

```
units value = lattice or box
    lattice = Xmax is defined in lattice units
    box = Xmax is defined in simulation box units
```

Examples:

```
fix 5 all dt/reset 10 1.0e−5 0.01 0.1
fix 5 all dt/reset 10 0.01 2.0 0.2 units box
```

Description:

Reset the timestep size every N steps during a run, so that no atom moves further than Xmax, based on current atom velocities and forces. This can be useful when starting from a configuration with overlapping atoms, where forces will be large. Or it can be useful when running an impact simulation where one or more high−energy atoms collide with a solid, causing a damage cascade.

This fix overrides the timestep size setting made by the timestep command. The new timestep size *dt* is computed in the following manner.

For each atom, the timestep is computed that would cause it to displace *Xmax* on the next integration step, as a function of its current velocity and force. Since performing this calculation exactly would require the solution to a quartic equation, a cheaper estimate is generated. The estimate is conservative in that the atom's displacement is guaranteed not to exceed *Xmax*, though it may be smaller.

Given this putative timestep for each atom, the minimum timestep value across all atoms is computed. Then the *Tmin* and *Tmax* bounds are applied, if specified. If one (or both) is specified as NULL, it is not applied.

When the run style is *respa*, this fix resets the outer loop (largest) timestep, which is the same timestep that the timestep command sets.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix.

This fix computes a global scalar and a global vector of length 1, which can be accessed by various output commands. The scalar is the current timestep size. The cumulative simulation time (in time units) is stored as the first element of the vector. The scalar and vector values calculated by this fix are "intensive".

No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions:

The cumulative time is zeroed when the fix is created and continuously accrues thereafter. Using the reset_timestep command while this fix is defined will mess up the time accumulation.

Related commands:

timestep

Default:

The option defaults is units = lattice.

fix efield command

Syntax:

```
fix ID group-ID efield ex ey ez
```

- ID, group−ID are documented in fix command
- efield = style name of this fix command
- ex,ey,ez = E−field component values (electric field units)
- any of ex,ey,ez can be a variable (see below)

Examples:

```
fix kick external-field efield 1.0 0.0 0.0
fix kick external-field efield 0.0 0.0 v_oscillate
```

Description:

Add a force F = qE to each charged atom in the group due to an external electric field being applied to the system.

Any of the 3 quantities defining the E−field components can be specified as an equal−style or atom−style variable, namely *ex*, *ey*, *ez*. If the value is a variable, it should be specified as v_name, where name is the variable name. In this case, the variable will be evaluated each timestep, and its value used to determine the E−field component.

Equal−style variables can specify formulas with various mathematical functions, and include thermo_style command keywords for the simulation box parameters and timestep and elapsed time. Thus it is easy to specify a time−dependent E−field.

Atom−style variables can specify the same formulas as equal−style variables but can also include per−atom values, such as atom coordinates. Thus it is easy to specify a spatially−dependent E−field with optional time−dependence as well.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix. No global or per−atom quantities are stored by this fix for access by various output commands. No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions: none

Related commands:

fix addforce

Default: none

fix enforce2d command

fix enforce2d/cuda command

Syntax:

```
fix ID group-ID enforce2d
```

- ID, group−ID are documented in fix command
- enforce2d = style name of this fix command

Examples:

```
fix 5 all enforce2d
```

Description:

Zero out the z−dimension velocity and force on each atom in the group. This is useful when running a 2d simulation to insure that atoms do not move from their initial z coordinate.

Styles with a *cuda* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in this section of the manual. The accelerated styles take the same arguments and should produce the same results, except for round−off and precision issues.

These accelerated styles are part of the "user−cuda" package. They are only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the −suffix command−line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See this section of the manual for more instructions on how to use the accelerated styles effectively.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix. No global or per−atom quantities are stored by this fix for access by various output commands. No parameter of this fix can be used with the *start/stop* keywords of the run command.

The forces due to this fix are imposed during an energy minimization, invoked by the minimize command.

Restrictions: none

Related commands: none

Default: none

fix evaporate command

Syntax:

```
fix ID group−ID evaporate N M region−ID seed
```

- ID, group−ID are documented in fix command
- evaporate = style name of this fix command
- N = delete atoms every this many timesteps
- M = number of atoms to delete each time
- region−ID = ID of region within which to perform deletions
- seed = random number seed to use for choosing atoms to delete
- zero or more keyword/value pairs may be appended

```
keyword = molecule
  molecule value = no or yes
```

Examples:

```
fix 1 solvent evaporate 1000 10 surface 49892
fix 1 solvent evaporate 1000 10 surface 38277 molecule yes
```

Description:

Remove M atoms from the simulation every N steps. This can be used, for example, to model evaporation of solvent particles or moleclues (i.e. drying) of a system. Every N steps, the number of atoms in the fix group and within the specifed region are counted. M of these are chosen at random and deleted. If there are less than M eligible particles, then all of them are deleted.

If the setting for the *molecule* keyword is *no*, then only single atoms are deleted. In this case, you should insure you do not delete only a portion of a molecule (only some of its atoms), or LAMMPS will soon generate an error when it tries to find those atoms. LAMMPS will warn you if any of the atoms eligible for deletion have a non−zero molecule ID, but does not check for this at the time of deletion.

If the setting for the *molecule* keyword is *yes*, then when an atom is chosen for deletion, the entire molecule it is part of is deleted. The count of deleted atoms is incremented by the number of atoms in the molecule, which may make it exceed *M*. If the molecule ID of the chosen atom is 0, then it is assumed to not be part of a molecule, and just the single atom is deleted.

As an example, if you wish to delete 10 water molecules every *N* steps, you should set *M* to 30. If only the water's oxygen atoms were in the fix group, then two hydrogen atoms would be deleted when an oxygen atom is selected for deletion, whether the hydrogens are inside the evaporation region or not.

Note that neighbor lists are re−built on timesteps that atoms are removed. Thus you should not remove atoms too frequently or you will incur overhead due to the cost of building neighbor lists.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix.

This fix computes a global scalar, which can be accessed by various output commands. The scalar is the cummulative number of deleted atoms. The scalar value calculated by this fix is "intensive".

No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions: none

Related commands:

fix deposit

Default:

The option defaults are molecule = no.

fix external command

Syntax:

```
fix ID group-ID external
```

- ID, group−ID are documented in fix command
- external = style name of this fix command

Examples:

```
fix 1 all external
```

Description:

This fix makes a callback each timestep or minimization iteration to an external driver program that is using LAMMPS as a library. This is a way to let another program compute forces on atoms which LAMMPS will include in its dynamics performed by the run command or its iterations performed by the minimize command

The callback function "foo" will be invoked every timestep or iteration as:

```
foo(ptr,timestep,nlocal,ids,x,fexternal);
```

which has this prototype:

void foo(void *, int, int, int *, double **, double **);

The arguments are as follows:

- ptr = pointer provided by and simply passed back to external driver
- timestep = current LAMMPS timestep
- nlocal = # of atoms on this processor
- ids = list of atom IDs on this processor
- x = coordinates of atoms on this processor
- fexternal = forces on atoms on this processor

Fexternal are the forces returned by the driver program, which LAMMPS adds to the current force on each atom.

See the couple/lammps_quest/lmpqst.cpp file in the LAMMPS distribution for an example of a coupling application that uses this fix, and how it makes a call to the fix to specify what function the fix should callback to. The sample application performs classical MD using quantum forces computed by a density functional code Quest.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix. No global or per−atom quantities are stored by this fix for access by various output commands. No parameter of this fix can be used with the *start/stop* keywords of the run command.

The forces due to this fix are imposed during an energy minimization, invoked by the minimize command.

However, LAMMPS knows nothing about the energy associated with these forces. So you should perform the minimization based on a force tolerance, not an energy tolerance.

Restrictions: none

Related commands: none

Default: none

fix freeze command

fix freeze/cuda command

Syntax:

```
fix ID group-ID freeze
```

- ID, group−ID are documented in fix command
- freeze = style name of this fix command

Examples:

```
fix 2 bottom freeze
```

Description:

Zero out the force and torque on a granular particle. This is useful for preventing certain particles from moving in a simulation. The granular pair styles also detect if this fix has been defined and compute interactions between frozen and non−frozen particles appropriately, as if the frozen particle has infinite mass.

Styles with a *cuda* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in this section of the manual. The accelerated styles take the same arguments and should produce the same results, except for round−off and precision issues.

These accelerated styles are part of the "user−cuda" package. They are only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the −suffix command−line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See this section of the manual for more instructions on how to use the accelerated styles effectively.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix.

This fix computes a global 3−vector of forces, which can be accessed by various output commands. This is the total force on the group of atoms before the forces on individual atoms are changed by the fix. The vector values calculated by this fix are "extensive".

No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions:

This fix is part of the "granular" package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

There can only be a single freeze fix defined. This is because other the granular pair styles treat frozen particles differently and need to be able to reference a single group to which this fix is applied.

Related commands: none

atom_style sphere

Default: none

fix gcmc command

Syntax:

```
fix ID group−ID gcmc N X M type seed T mu displace keyword values ...
```

- ID, group−ID are documented in fix command
- gcmc = style name of this fix command
- N = invoke this fix every N steps
- X = number of exchanges to attempt every N steps
- M = number of MC displacements to attempt every N steps
- type = atom type of exchanged particles
- seed = random # seed (positive integer)
- T = temperature of the ideal gas reservoir (temperature units)
- mu = chemical potential of the ideal gas reservoir (energy units)
- displace = maximum Monte Carlo displacement distance (length units)
- zero or more keyword/value pairs may be appended to args

```
keyword = molecule
  molecule value = no or yes
```

Examples:

```
fix 2 all gcmc 10 1000 1000 2 29494 298.0 −0.5 0.01
fix 3 all gcmc 10 100 100 1 3456543 3.0 −2.5 0.1 molecule yes
```

Description:

This fix performs grand canonical Monte Carlo (GCMC) exchanges of particles of the given type with an imaginary ideal gas reservoir at the specified T and chemical potential (mu) as discussed in (Frenkel). If used with the fix nvt command, simulations in the grand canonical enemble (muVT, constant chemical potential, constant volume, and constant temperature) can be performed. Specific uses include computing isotherms in microporous materials, or computing vapor−liquid coexistence curves.

Perform up to X exchanges of particles of the given type between the simulation domain and the imaginary reservoir every N timesteps. Also perform M Monte Carlo displacements of particles of the given type within the simulation domain. M should typically be chosen to be approximately equal to the expected number of particles of the given type within the domain, which will result in roughly one MC translation per particle per MC cycle.

This fix cannot be used to perform MC displacements of particles other than the exchanged type. All particles in the simulation domain can be moved using regular time integration displacements, e.g. via fix_nvt, resulting in a hybrid GCMC+MD simulation.

If used with fix_nvt, the temperature of the imaginary reservoir, T, should be set to be equivalent to the target temperature used in fix_nvt. Otherwise, the imaginary reservoir will not be in thermal equilibrium with the simulation domain.

Note that neighbor lists are re−built every timestep that this fix is invoked, so you should not set N to be too small. However, periodic rebuilds are necessary in order to avoid dangerous rebuilds and missed interactions. Specifically, avoid performing so many MC displacements per timestep that a particle can move beyond the neighbor list skin distance. See the neighbor command for details.

When a particle is to be inserted, its coordinates are chosen as a random position within the current simulation domain, and its velocity is randomly chosen from the specified temperature distribution given by T.

Exchanged particles have the specified atom type and are assigned to two groups: the default group "all" and the group specified in the fix gcmc command (which can also be "all").

If the setting for the *molecule* keyword is *no*, then only single atoms are exchanged. In this case, you should ensure you do not delete only a portion of a molecule (only some of its atoms), or LAMMPS will soon generate an error when it tries to find those atoms. LAMMPS will warn you if any of the atoms eligible for deletion have a non−zero molecule ID, but does not check for this at the time of deletion.

If the setting for the *molecule* keyword is *yes*, entire molecules are exchanged. This feature is not yet supported.

Use of this fix typically will cause the number of atoms to fluctuate, therefore, you will want to use the compute_modify command to insure that the current number of atoms is used as a normalizing factor each time temperature is computed. Here is the necessary command:

```
compute_modify thermo_temp dynamic yes
```

If LJ units are used, note that a value of 0.18292026 is used by this fix as the reduced value for Planck's constant. This value was derived from LJ paramters for argon, where h* = h/sqrt(sigma^2 * epsilon * mass), sigma = 3.429 angstroms, epsilon/k = 121.85 K, and mass = 39.948 amu.

Restart, fix_modify, output, run start/stop, minimize info:

This fix writes the state of the deposition to binary restart files. This includes information about the random number generator seed, the next timestep for MC exchanges, etc. See the read_restart command for info on how to re−specify a fix in an input script that reads a restart file, so that the operation of the fix continues in an uninterrupted fashion.

None of the fix_modify options are relevant to this fix.

This fix computes a global vector of length 6, which can be accessed by various output commands. The vector values are the following global cummulative quantities:

- 1 = displacement attempts
- 2 = displacement successes
- 3 = deletion attempts
- 4 = deletion successes
- 5 = insertion attempts
- 6 = insertion successes

The vector values calculated by this fix are "extensive".

No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions:

This fix is part of the "mc" package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

Do not set "neigh_modify once yes" or else this fix will never be called. Reneighboring is required.

You cannot currently exchange charged particles or molecules with a net charge.

Only pairwise interactions, as defined by the pair_style command, are included in this calculation. Long−range interactions due to a kspace_style command are not included. Not all pair potentials can be evaluated in a pairwise mode as required by this fix. For example, 3−body potentials, such as Tersoff and Stillinger−Weber cannot be used. EAM potentials for metals only include the pair potential portion of the EAM interaction, not the embedding term.

Related commands:

fix_nvt, neighbor, fix_deposit, fix_evaporate

Default:

The option defaults are molecule = no.

(Frenkel) Frenkel and Smit, Understanding Molecular Simulation, Academic Press, London, 2002.

fix gravity command

fix gravity/cuda command

Syntax:

```
fix ID group gravity style magnitude args
```

- ID, group are documented in fix command
- gravity = style name of this fix command
- magnitude = size of acceleration (force/mass units)
- style = *chute* or *spherical* or *gradient* or *vector*

```
  chute args = angle
      angle = angle in +x away from −z or −y axis in 3d/2d (in degrees)
    spherical args = phi theta
      phi = azimuthal angle from +x axis (in degrees)
      theta = angle from +z or +y axis in 3d/2d (in degrees)
    gradient args = phi theta phi_grad theta_grad
      phi = azimuthal angle from +x axis (in degrees)
      theta = angle from +z or +y axis in 3d/2d (in degrees)
      phi_grad = rate of change of angle phi (full rotations per time unit)
      theta_grad = rate of change of angle theta (full rotations per time unit)
    vector args = x y z
      x y z = vector direction to apply the acceleration
```

Examples:

```
fix 1 all gravity 1.0 chute 24.0
fix 1 all gravity 1.0 spherical 0.0 −180.0
fix 1 all gravity 1.0 gradient 0.0 −180.0 0.0 0.1
fix 1 all gravity 100.0 vector 1 1 0
```

Description:

Impose an additional acceleration on each particle in the group. This fix is typically used with granular systems to include a "gravity" term acting on the macroscopic particles. More generally, it can represent any kind of driving field, e.g. a pressure gradient inducing a Poiseuille flow in a fluid. Note that this fix operates differently than the fix addforce command. The addforce fix adds the same force to each atom, independent of its mass. This command imparts the same acceleration to each atom (force/mass).

The *magnitude* of the acceleration is specified in force/mass units. For granular systems (LJ units) this is typically 1.0. See the units command for details.

Style *chute* is typically used for simulations of chute flow where the specified angle is the chute angle, with flow occurring in the +x direction. For 3d systems, the tilt is away from the z axis; for 2d systems, the tilt is away from the y axis.

Style *spherical* allows an arbitrary 3d direction to be specified for the acceleration vector. Phi and theta are defined in the usual spherical coordinates. Thus for acceleration acting in the −z direction, theta would be 180.0 (or −180.0). Theta = 90.0 and phi = −90.0 would mean acceleration acts in the −y direction. For 2d systems, *phi* is ignored and *theta* is an angle in the xy plane where theta = 0.0 is the y−axis.

Style *gradient* is the same as style *spherical* except that the direction of the acceleration vector is time dependent. The units of the gradient arguments are in full rotations per time unit. E.g. a timestep of 0.001 and a gradient of 0.1 means the acceleration vector would rotate thru 360 degrees every 10,000 timesteps. For the time−dependent case, the initial direction of the acceleration vector is set to phi,theta when the fix is specified and evolves thereafter. For 2d systems, *phi* and *phi_grad* are ignored.

Style *vector* imposes an acceleration in the vector direction given by (x,y,z). For 2d systems, the z component is ignored.

Styles with a *cuda* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in this section of the manual. The accelerated styles take the same arguments and should produce the same results, except for round−off and precision issues.

These accelerated styles are part of the "user−cuda" package. They are only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the −suffix command−line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See this section of the manual for more instructions on how to use the accelerated styles effectively.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix. No global or per−atom quantities are stored by this fix for access by various output commands. No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions: none

Related commands:

atom_style sphere, fix addforce

Default: none

fix heat command

Syntax:

```
fix ID group-ID heat N eflux
```

- ID, group−ID are documented in fix command
- heat = style name of this fix command
- N = add/subtract heat every this many timesteps
- eflux = rate of heat addition or subtraction (energy/time units)
- zero or more keyword/value pairs may be appended to args
- keyword = *region*

```
region value = region-ID
    region-ID = ID of region atoms must be in to have added force
```

Examples:

```
fix 3 qin heat 1 1.0
fix 4 qout heat 1 -1.0 region top
```

Description:

Add non−translational kinetic energy (heat) to a group of atoms such that their aggregate momentum is conserved. Two of these fixes can be used to establish a temperature gradient across a simulation domain by adding heat (energy) to one group of atoms (hot reservoir) and subtracting heat from another (cold reservoir). E.g. a simulation sampling from the McDLT ensemble.

If the *region* keyword is used, the atom must be in both the group and the specified geometric region in order to have energy added or subtracted to it. If not specified, then the atoms in the group are affected wherever they may move to.

Heat addition/subtraction is performed every N timesteps. The *eflux* parameter determines the change in aggregate energy of the entire group of atoms per unit time, e.g. in eV/psec for metal units. Thus it is an "extensive" quantity, meaning its magnitude should be scaled with the number of atoms in the group. Since *eflux* is independent of N or the timestep, a larger value of N will add/subtract a larger amount of energy each time the fix is invoked. If heat is subtracted from the system too aggressively so that the group's kinetic energy would go to zero, LAMMPS halts with an error message.

Fix heat is different from a thermostat such as fix nvt or fix temp/rescale in that energy is added/subtracted continually. Thus if there isn't another mechanism in place to counterbalance this effect, the entire system will heat or cool continuously. You can use multiple heat fixes so that the net energy change is 0.0 or use fix viscous to drain energy from the system.

This fix does not change the coordinates of its atoms; it only scales their velocities. Thus you must still use an integration fix (e.g. fix nve) on the affected atoms. This fix should not normally be used on atoms that have their temperature controlled by another fix − e.g. fix nvt or fix langevin fix.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix.

This fix computes a global scalar which can be accessed by various output commands. This scalar is the most recent value by which velocites were scaled. The scalar value calculated by this fix is "intensive".

No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions: none

Related commands:

compute temp, compute temp/region

Default: none

fix imd command

Syntax:

```
fix ID group-ID imd trate port keyword values ...
```

- ID, group−ID are documented in fix command
- imd = style name of this fix command
- port = port number on which the fix listens for an IMD client
- keyword = *unwrap* or *fscale* or *trate*

```
  unwrap arg = on or off
      off = coordinates are wrapped back into the principal unit cell (default)
      on = "unwrapped" coordinates using the image flags used
    fscale arg = factor
      factor = floating point number to scale IMD forces (default: 1.0)
    trate arg = transmission rate of coordinate data sets (default: 1)
    nowait arg = on or off
      off = LAMMPS waits to be connected to an IMD client before continuing (default)
      on = LAMMPS listens for an IMD client, but continues with the run
```

Examples:

```
fix vmd all imd 5678
fix comm all imd 8888 trate 5 unwrap on fscale 10.0
```

Description:

This fix implements the "Interactive MD" (IMD) protocol which allows realtime visualization and manipulation of MD simulations through the IMD protocol, as initially implemented in VMD and NAMD. Specifically it allows LAMMPS to connect an IMD client, for example the VMD visualization program, so that it can monitor the progress of the simulation and interactively apply forces to selected atoms.

If LAMMPS is compiled with the preprocessor flag −DLAMMPS_ASYNC_IMD then fix imd will use posix threads to spawn a thread on MPI rank 0 in order to offload data reading and writing from the main execution thread and potentiall lower the inferred latencies for slow communication links. This feature has only been tested under linux.

There are example scripts for using this package with LAMMPS in examples/USER/imd. Additional examples and a driver for use with the Novint Falcon game controller as haptic device can be found at: http://sites.google.com/site/akohlmey/software/vrpn−icms.

The source code for this fix includes code developed by the Theoretical and Computational Biophysics Group in the Beckman Institute for Advanced Science and Technology at the University of Illinois at Urbana−Champaign. We thank them for providing a software interface that allows codes like LAMMPS to hook to VMD.

Upon initialization of the fix, it will open a communication port on the node with MPI task 0 and wait for an incoming connection. As soon as an IMD client is connected, the simulation will continue and the fix will send the current coordinates of the fix's group to the IMD client at every trate MD step. When using r−RESPA, trate applies to the steps of the outmost RESPA level. During a run with an active IMD connection also the IMD client can request to apply forces to selected atoms of the fix group.

The port number selected must be an available network port number. On many machines, port numbers < 1024 are reserved for accounts with system manager privilege and specific applications. If multiple imd fixes would be active at the same time, each needs to use a different port number.

The *nowait* keyword controls the behavior of the fix when no IMD client is connected. With the default setting of *off*, LAMMPS will wait until a connection is made before continuing with the execution. Setting *nowait* to *on* will have the LAMMPS code be ready to connect to a client, but continue with the simulation. This can for example be used to monitor the progress of an ongoing calculation without the need to be permanently connected or having to download a trajectory file.

The *trate* keyword allows to select how often the coordinate data is sent to the IMD client. It can also be changed on request of the IMD client through an IMD protocol message. The *unwrap* keyword allows to send "unwrapped" coordinates to the IMD client that undo the wrapping back of coordinates into the principle unit cell, as done by default in LAMMPS. The *fscale* keyword allows to apply a scaling factor to forces transmitted by the IMD client. The IMD protocols stipulates that forces are transferred in kcal/mol/angstrom under the assumption that coordinates are given in angstrom. For LAMMPS runs with different units or as a measure to tweak the forces generated by the manipulation of the IMD client, this option allows to make adjustments.

To connect VMD to a listening LAMMPS simulation on the same machine with fix imd enabled, one needs to start VMD and load a coordinate or topology file that matches the fix group. When the VMD command prompts appears, one types the command line:

```
imd connect localhost 5678
```

This assumes that *fix imd* was started with 5678 as a port number for the IMD protocol.

The steps to do interactive manipulation of a running simulation in VMD are the following:

In the Mouse menu of the VMD Main window, select "Mouse −> Force −> Atom". You may alternately select "Residue", or "Fragment" to apply forces to whole residues or fragments. Your mouse can now be used to apply forces to your simulation. Click on an atom, residue, or fragment and drag to apply a force. Click quickly without moving the mouse to turn the force off. You can also use a variety of 3D position trackers to apply forces to your simulation. Game controllers or haptic devices with force−feedback such as the Novint Falcon or Sensable PHANTOM allow you to feel the resistance due to inertia or interactions with neighbors that the atoms experience you are trying to move, as if they were real objects. See the VMD IMD Homepage and the VRPN−ICMS Homepage for more details.

If IMD control messages are received, a line of text describing the message and its effect will be printed to the LAMMPS output screen, if screen output is active.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix. No global scalar or vector or per−atom quantities are stored by this fix for access by various output commands. No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions:

This fix is part of the "user−misc" package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

On platforms that support multi−threading, this fix can be compiled in a way that the coordinate transfers to the IMD client can be handled from a separate thread, when LAMMPS is compiled with the −DLAMMPS_ASYNC_IMD preprocessor flag. This should to keep MD loop times low and transfer rates high, especially for systems with many atoms and for slow connections.

When used in combination with VMD, a topology or coordinate file has to be loaded, which matches (in number and ordering of atoms) the group the fix is applied to. The fix internally sorts atom IDs by ascending integer value; in VMD (and thus the IMD protocol) those will be assigned 0−based consecutive index numbers.

When using multiple active IMD connections at the same time, each needs to use a different port number.

Related commands: none

Default: none

fix indent command

Syntax:

```
fix ID group−ID indent K keyword values ...
```

- ID, group−ID are documented in fix command
- indent = style name of this fix command
- K = force constant for indenter surface (force/distance^2 units)
- one or more keyword/value pairs may be appended
- keyword = *sphere* or *cylinder* or *plane* or *side* or *units*

```
  sphere args = x y z R
      x,y,z = initial position of center of indenter (distance units)
      R = sphere radius of indenter (distance units)
      any of x,y,z,R can be a variable (see below)
    cylinder args = dim c1 c2 R
      dim = x or y or z = axis of cylinder
      c1,c2 = coords of cylinder axis in other 2 dimensions (distance units)
      R = cylinder radius of indenter (distance units)
      any of c1,c2,R can be a variable (see below)
    plane args = dim pos side
      dim = x or y or z = plane perpendicular to this dimension
      pos = position of plane in dimension x, y, or z (distance units)
      pos can be a variable (see below)
      side = lo or hi
    side value = in or out
      in = the indenter acts on particles inside the sphere or cylinder
      out = the indenter acts on particles outside the sphere or cylinder
    units value = lattice or box
      lattice = the geometry is defined in lattice units
      box = the geometry is defined in simulation box units
```

Examples:

```
fix 1 all indent 10.0 sphere 0.0 0.0 15.0 3.0
fix 1 all indent 10.0 sphere v_x v_y 0.0 v_radius side in
fix 2 flow indent 10.0 cylinder z 0.0 0.0 10.0 units box
```

Description:

Insert an indenter within a simulation box. The indenter repels all atoms that touch it, so it can be used to push into a material or as an obstacle in a flow. Or it can be used as a constraining wall around a simulation; see the discussion of the *side* keyword below.

The indenter can either be spherical or cylindrical or planar. You must set one of those 3 keywords.

A spherical indenter exerts a force of magnitude

```
F(r) = − K (r − R)^2
```

on each atom where K is the specified force constant, r is the distance from the atom to the center of the indenter, and R is the radius of the indenter. The force is repulsive and F(r) = 0 for $r > R$.

A cylindrical indenter exerts the same force, except that *r* is the distance from the atom to the center axis of the cylinder. The cylinder extends infinitely along its axis.

Spherical and cylindrical indenters account for periodic boundaries in two ways. First, the center point of a spherical indenter (x,y,z) or axis of a cylindrical indenter (c1,c2) is remapped back into the simulation box, if the box is periodic in a particular dimension. This occurs every timestep if the indenter geometry is specified with a variable (see below), e.g. it is moving over time. Second, the calculation of distance to the indenter center or axis accounts for periodic boundaries. Both of these mean that an indenter can effectively move through and straddle one or more periodic boundaries.

A planar indenter is really an axis−aligned infinite−extent wall exerting the same force on atoms in the system, where *R* is the position of the plane and *r*−*R* is the distance from the plane. If the *side* parameter of the plane is specified as *lo* then it will indent from the lo end of the simulation box, meaning that atoms with a coordinate less than the plane's current position will be pushed towards the hi end of the box and atoms with a coordinate higher than the plane's current position will feel no force. Vice versa if *side* is specified as *hi*.

Any of the 4 quantities defining a spherical indenter's geometry can be specified as an equal−style variable, namely *x*, *y*, *z*, or *R*. Similarly, for a cylindrical indenter, any of *c1*, *c2*, or *R*, can be a variable. For a planar indenter, *pos* can be a variable. If the value is a variable, it should be specified as v_name, where name is the variable name. In this case, the variable will be evaluated each timestep, and its value used to define the indenter geometry.

Note that equal−style variables can specify formulas with various mathematical functions, and include thermo_style command keywords for the simulation box parameters and timestep and elapsed time. Thus it is easy to specify indenter properties that change as a function of time or span consecutive runs in a continuous fashion. For the latter, see the *start* and *stop* keywords of the run command and the *elaplong* keyword of thermo_style custom for details.

For example, if a spherical indenter's x−position is specfied as v_x, then this variable definition will keep it's center at a relative position in the simulation box, 1/4 of the way from the left edge to the right edge, even if the box size changes:

```
variable x equal "xlo + 0.25*lx"
```

Similarly, either of these variable definitions will move the indenter from an initial position at 2.5 at a constant velocity of 5:

```
variable x equal "2.5 + 5*elaplong*dt"
variable x equal vdisplace(2.5,5)
```

If a spherical indenter's radius is specified as v_r, then these variable definitions will grow the size of the indenter at a specfied rate.

```
variable r0 equal 0.0
variable rate equal 1.0
variable r equal "v_r0 + step*dt*v_rate"
```

If the *side* keyword is specified as *out*, which is the default, then particles outside the indenter are pushded away from its outer surface, as described above. This only applies to spherical or cylindrical indenters. If the *side* keyword is specified as *in*, the action of the indenter is reversed. Particles inside the indenter are pushed away from its inner surface. In other words, the indenter is now a containing wall that traps the particles inside it. If the radius shrinks over time, it will squeeze the particles.

The *units* keyword determines the meaning of the distance units used to define the indenter geometry. A *box* value selects standard distance units as defined by the units command, e.g. Angstroms for units = real or metal. A *lattice* value means the distance units are in lattice spacings. The lattice command must have been previously used to define the lattice spacing. The (x,y,z) coords of the indenter position are scaled by the x,y,z lattice spacings respectively. The radius of a spherical or cylindrical indenter is scaled by the x lattice spacing.

Note that the units keyword only affects indenter geometry parameters specified directly with numbers, not those specified as variables. In the latter case, you should use the *xlat*, *ylat*, *zlat* keywords of the thermo_style command if you want to include lattice spacings in a variable formula.

The force constant *K* is not affected by the *units* keyword. It is always in force/distance^2 units where force and distance are defined by the units command. If you wish K to be scaled by the lattice spacing, you can define K with a variable whose formula contains *xlat*, *ylat*, *zlat* keywords of the thermo_style command, e.g.

```
variable k equal 100.0/xlat/xlat
fix 1 all indent $k sphere ...
```

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files.

The fix_modify *energy* option is supported by this fix to add the energy of interaction between atoms and the indenter to the system's potential energy as part of thermodynamic output. The energy of each particle interacting with the indenter is K/3 (r − R)^3.

This fix computes a global scalar energy and a global 3−vector of forces (on the indenter), which can be accessed by various output commands. The scalar and vector values calculated by this fix are "extensive".

The forces due to this fix are imposed during an energy minimization, invoked by the minimize command. Note that if you define the indenter geometry with a variable using a time−dependent formula, LAMMPS uses the iteration count in the minimizer as the timestep. But it is almost certainly a bad idea to have the indenter change its position or size during a minimization. LAMMPS does not check if you have done this.

IMPORTANT NOTE: If you want the atom/indenter interaction energy to be included in the total potential energy of the system (the quantity being minimized), you must enable the fix_modify *energy* option for this fix.

Restrictions: none

Related commands: none

Default:

The option defaults are side = out and units = lattice.

fix langevin command

Syntax:

```
fix ID group-ID langevin Tstart Tstop damp seed keyword values ...
```

- ID, group−ID are documented in fix command
- langevin = style name of this fix command
- Tstart,Tstop = desired temperature at start/end of run (temperature units)
- damp = damping parameter (time units)
- seed = random number seed to use for white noise (positive integer)
- zero or more keyword/value pairs may be appended
- keyword = *angmom* or *omega* or *scale* or *tally* or *zero*

```
  angmom value = no or yes
      no = do not thermostat rotational degrees of freedom via the angular momentum
      yes = do thermostat rotational degrees of freedom via the angular momentum
    omega value = no or yes
      no = do not thermostat rotational degrees of freedom via then angular velocity
      yes = do thermostat rotational degrees of freedom via the angular velocity
    scale values = type ratio
      type = atom type (1-N)
      ratio = factor by which to scale the damping coefficient
    tally value = no or yes
      no = do not tally the energy added/subtracted to atoms
      yes = do tally the energy added/subtracted to atoms
    zero value = no or yes
      no = do not set total random force to zero
      yes = set total random force to zero
```

Examples:

```
fix 3 boundary langevin 1.0 1.0 1000.0 699483
fix 1 all langevin 1.0 1.1 100.0 48279 scale 3 1.5
```

Description:

Apply a Langevin thermostat as described in (Schneider) to a group of atoms which models an interaction with a background implicit solvent. Used with fix nve, this command performs Brownian dynamics (BD), since the total force on each atom will have the form:

```
F = Fc + Ff + Fr
Ff = - (m / damp) v
Fr is proportional to sqrt(Kb T m / (dt damp))
```

Fc is the conservative force computed via the usual inter−particle interactions (pair_style, bond_style, etc).

The Ff and Fr terms are added by this fix on a per−particle basis. See the pair_style dpd/tstat command for a thermostatting option that adds similar terms on a pairwise basis to pairs of interacting particles.

Ff is a frictional drag or viscous damping term proportional to the particle's velocity. The proportionality constant for each atom is computed as m/damp, where m is the mass of the particle and damp is the damping factor specified by the user.

Fr is a force due to solvent atoms at a temperature T randomly bumping into the particle. As derived from the fluctuation/dissipation theorem, its magnitude as shown above is proportional to sqrt(Kb T m / dt damp), where Kb is the Boltzmann constant, T is the desired temperature, m is the mass of the particle, dt is the timestep size, and damp is the damping factor. Random numbers are used to randomize the direction and magnitude of this force as described in (Dunweg), where a uniform random number is used (instead of a Gaussian random number) for speed.

Note that the thermostat effect of this fix is applied to only the translational degrees of freedom for the particles, which is an important consideration if extended spherical or aspherical particles which have rotational degrees of freedom are being thermostatted with this fix. The translational degrees of freedom can also have a bias velocity removed from them before thermostatting takes place; see the description below.

IMPORTANT NOTE: Unlike the fix nvt command which performs Nose/Hoover thermostatting AND time integration, this fix does NOT perform time integration. It only modifies forces to effect thermostatting. Thus you must use a separate time integration fix, like fix nve to actually update the velocities and positions of atoms using the modified forces. Likewise, this fix should not normally be used on atoms that also have their temperature controlled by another fix − e.g. by fix nvt or fix temp/rescale commands.

See this howto section of the manual for a discussion of different ways to compute temperature and perform thermostatting.

The desired temperature at each timestep is a ramped value during the run from *Tstart* to *Tstop*.

Like other fixes that perform thermostatting, this fix can be used with compute commands that remove a "bias" from the atom velocities. E.g. removing the center−of−mass velocity from a group of atoms or removing the x−component of velocity from the calculation. This is not done by default, but only if the fix_modify command is used to assign a temperature compute to this fix that includes such a bias term. See the doc pages for individual compute commands to determine which ones include a bias. In this case, the thermostat works in the following manner: bias is removed from each atom, thermostatting is performed on the remaining thermal degrees of freedom, and the bias is added back in.

The *damp* parameter is specified in time units and determines how rapidly the temperature is relaxed. For example, a value of 100.0 means to relax the temperature in a timespan of (roughly) 100 time units (tau or fmsec or psec − see the units command). The damp factor can be thought of as inversely related to the viscosity of the solvent. I.e. a small relaxation time implies a hi−viscosity solvent and vice versa. See the discussion about gamma and viscosity in the documentation for the fix viscous command for more details.

The random # *seed* must be a positive integer. A Marsaglia random number generator is used. Each processor uses the input seed to generate its own unique seed and its own stream of random numbers. Thus the dynamics of the system will not be identical on two runs on different numbers of processors.

The keyword/value option pairs are used in the following ways.

The keyword *angmom* and *omega* keywords enable thermostatting of rotational degrees of freedom in addition to the usual translational degrees of freedom. This can only be done for finite−size particles. A simulation using atom_style sphere defines an omega for finite−size spheres. A simulation using atom_style ellipsoid defines a finite size and shape for aspherical particles and an angular momentum. The Langevin formulas for thermostatting the rotational degrees of freedom are the same as those above, where force is replaced by torque, m is replaced by the moment of inertia I, and v is replaced by omega (which is derived from the angular momentum in the case of aspherical particles). The rotational temperature of the particles can be monitored by the compute temp/sphere and compute temp/asphere commands with their rotate options.

The keyword *scale* allows the damp factor to be scaled up or down by the specified factor for atoms of that type. This can be useful when different atom types have different sizes or masses. It can be used multiple times to adjust damp for several atom types. Note that specifying a ratio of 2 increases the relaxation time which is equivalent to the solvent's viscosity acting on particles with 1/2 the diameter. This is the opposite effect of scale factors used by the fix viscous command, since the damp factor in fix *langevin* is inversely related to the gamma factor in fix *viscous*. Also note that the damping factor in fix *langevin* includes the particle mass in Ff, unlike fix *viscous*. Thus the mass and size of different atom types should be accounted for in the choice of ratio values.

The keyword *tally* enables the calculation of the cumulative energy added/subtracted to the atoms as they are thermostatted. Effectively it is the energy exchanged between the infinite thermal reservoir and the particles. As described below, this energy can then be printed out or added to the potential energy of the system to monitor energy conservation.

The keyword *zero* can be used to eliminate drift due to the thermostat. Because the random forces on different atoms are independent, they do not sum exactly to zero. As a result, this fix applies a small random force to the entire system, and the center−of−mass of the system undergoes a slow random walk. If the keyword *zero* is set to *yes*, the total random force is set exactly to zero by subtracting off an equal part of it from each atom in the group. As a result, the center−of−mass of a system with zero initial momentum will not drift over time.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. Because the state of the random number generator is not saved in restart files, this means you cannot do "exact" restarts with this fix, where the simulation continues on the same as if no restart had taken place. However, in a statistical sense, a restarted simulation should produce the same behavior.

The fix_modify *temp* option is supported by this fix. You can use it to assign a temperature compute you have defined to this fix which will be used in its thermostatting procedure, as described above. For consistency, the group used by this fix and by the compute should be the same.

The fix_modify *energy* option is supported by this fix to add the energy change induced by Langevin thermostatting to the system's potential energy as part of thermodynamic output. Note that use of this option requires setting the *tally* keyword to *yes*.

This fix computes a global scalar which can be accessed by various output commands. The scalar is the cummulative energy change due to this fix. The scalar value calculated by this fix is "extensive". Note that calculation of this quantity requires setting the *tally* keyword to *yes*.

This fix can ramp its target temperature over multiple runs, using the *start* and *stop* keywords of the run command. See the run command for details of how to do this.

This fix is not invoked during energy minimization.

Restrictions: none

Related commands:

fix nvt, fix temp/rescale, fix viscous, fix nvt, pair_style dpd/tstat

Default:

The option defaults are angmom = no, omega = no, scale = 1.0 for all types, tally = no, zero = no.

(Dunweg) Dunweg and Paul, Int J of Modern Physics C, 2, 817−27 (1991).

(Schneider) Schneider and Stoll, Phys Rev B, 17, 1302 (1978).

fix langevin/eff command

Syntax:

```
fix ID group−ID langevin/eff Tstart Tstop damp seed keyword values ...
```

- ID, group−ID are documented in fix command
- langevin/eff = style name of this fix command
- Tstart,Tstop = desired temperature at start/end of run (temperature units)
- damp = damping parameter (time units)
- seed = random number seed to use for white noise (positive integer)
- zero or more keyword/value pairs may be appended

```
keyword = scale or tally
  scale values = type ratio
    type = atom type (1−N)
    ratio = factor by which to scale the damping coefficient
  tally values = no or yes
    no = do not tally the energy added/subtracted to atoms
    yes = do tally the energy added/subtracted to atoms
```

Examples:

```
fix 3 boundary langevin/eff 1.0 1.0 10.0 699483
fix 1 all langevin/eff 1.0 1.1 10.0 48279 scale 3 1.5
```

Description:

Apply a Langevin thermostat as described in (Schneider) to a group of nuclei and electrons in the electron force field model. Used with fix nve/eff, this command performs Brownian dynamics (BD), since the total force on each atom will have the form:

```
F = Fc + Ff + Fr
Ff = − (m / damp) v
Fr is proportional to sqrt(Kb T m / (dt damp))
```

Fc is the conservative force computed via the usual inter−particle interactions (pair_style).

The Ff and Fr terms are added by this fix on a per−particle basis.

The operation of this fix is exactly like that described by the fix langevin command, except that the thermostatting is also applied to the radial electron velocity for electron particles.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. Because the state of the random number generator is not saved in restart files, this means you cannot do "exact" restarts with this fix, where the simulation continues on the same as if no restart had taken place. However, in a statistical sense, a restarted simulation should produce the same behavior.

The fix_modify *temp* option is supported by this fix. You can use it to assign a temperature compute you have defined to this fix which will be used in its thermostatting procedure, as described above. For consistency, the

group used by this fix and by the compute should be the same.

The fix_modify *energy* option is supported by this fix to add the energy change induced by Langevin thermostatting to the system's potential energy as part of thermodynamic output. Note that use of this option requires setting the *tally* keyword to *yes*.

This fix computes a global scalar which can be accessed by various output commands. The scalar is the cummulative energy change due to this fix. The scalar value calculated by this fix is "extensive". Note that calculation of this quantity requires setting the *tally* keyword to *yes*.

This fix can ramp its target temperature over multiple runs, using the *start* and *stop* keywords of the run command. See the run command for details of how to do this.

This fix is not invoked during energy minimization.

Restrictions: none

This fix is part of the "user−eff" package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

Related commands:

fix langevin

Default:

The option defaults are scale = 1.0 for all types and tally = no.

(Dunweg) Dunweg and Paul, Int J of Modern Physics C, 2, 817−27 (1991).

(Schneider) Schneider and Stoll, Phys Rev B, 17, 1302 (1978).

fix lineforce command

Syntax:

```
fix ID group-ID lineforce x y z
```

- ID, group−ID are documented in fix command
- lineforce = style name of this fix command
- x y z = direction of line as a 3−vector

Examples:

```
fix hold boundary lineforce 0.0 1.0 1.0
```

Description:

Adjust the forces on each atom in the group so that only the component of force along the linear direction specified by the vector (x,y,z) remains. This is done by subtracting out components of force in the plane perpendicular to the line.

If the initial velocity of the atom is 0.0 (or along the line), then it should continue to move along the line thereafter.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix. No global or per−atom quantities are stored by this fix for access by various output commands. No parameter of this fix can be used with the *start/stop* keywords of the run command.

The forces due to this fix are imposed during an energy minimization, invoked by the minimize command.

Restrictions: none

Related commands:

fix planeforce

Default: none

fix_modify command

Syntax:

```
fix_modify fix-ID keyword value ...
```

- fix−ID = ID of the fix to modify
- one or more keyword/value pairs may be appended
- keyword = *temp* or *press* or *energy*

```
  temp value = compute ID that calculates a temperature
    press value = compute ID that calculates a pressure
    energy value = yes or no
```

Examples:

```
fix_modify 3 temp myTemp press myPress
fix_modify 1 energy yes
```

Description:

Modify one or more parameters of a previously defined fix. Only specific fix styles support specific parameters. See the doc pages for individual fix commands for info on which ones support which fix_modify parameters.

The *temp* keyword is used to determine how a fix computes temperature. The specified compute ID must have been previously defined by the user via the compute command and it must be a style of compute that calculates a temperature. All fixes that compute temperatures define their own compute by default, as described in their documentation. Thus this option allows the user to override the default method for computing T.

The *press* keyword is used to determine how a fix computes pressure. The specified compute ID must have been previously defined by the user via the compute command and it must be a style of compute that calculates a pressure. All fixes that compute pressures define their own compute by default, as described in their documentation. Thus this option allows the user to override the default method for computing P.

For fixes that calculate a contribution to the potential energy of the system, the *energy* keyword will include that contribution in thermodynamic output of potential energy. See the thermo_style command for info on how potential energy is output. The contribution by itself can be printed by using the keyword f_ID in the thermo_style custom command, where ID is the fix−ID of the appropriate fix. Note that you must use this setting for a fix if you are using it when performing an energy minimization and if you want the energy and forces it produces to be part of the optimization criteria.

Restrictions: none

Related commands:

fix, compute temp, compute pressure, thermo_style

Default:

The option defaults are temp = ID defined by fix, press = ID defined by fix, energy = no.

fix momentum command

Syntax:

```
fix ID group-ID momentum N keyword values ...
```

- ID, group−ID are documented in fix command
- momentum = style name of this fix command
- N = adjust the momentum every this many timesteps one or more keyword/value pairs may be appended
- keyword = *linear* or *angular*

```
  linear values = xflag yflag zflag
      xflag,yflag,zflag = 0/1 to exclude/include each dimension
    angular values = none
```

Examples:

```
fix 1 all momentum 1 linear 1 1 0
fix 1 all momentum 100 linear 1 1 1 angular
```

Description:

Zero the linear and/or angular momentum of the group of atoms every N timesteps by adjusting the velocities of the atoms. One (or both) of the *linear* or *angular* keywords must be specified.

If the *linear* keyword is used, the linear momentum is zeroed by subtracting the center−of−mass velocity of the group from each atom. This does not change the relative velocity of any pair of atoms. One or more dimensions can be excluded from this operation by setting the corresponding flag to 0.

If the *angular* keyword is used, the angular momentum is zeroed by subtracting a rotational component from each atom.

This command can be used to insure the entire collection of atoms (or a subset of them) does not drift or rotate during the simulation due to random perturbations (e.g. fix langevin thermostatting).

Note that the velocity command can be used to create initial velocities with zero aggregate linear and/or angular momentum.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix. No global or per−atom quantities are stored by this fix for access by various output commands. No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions: none

Related commands:

fix recenter, velocity

Default: none

fix move command

Syntax:

```
fix ID group-ID move style args keyword values ...
```

- ID, group−ID are documented in fix command
- move = style name of this fix command
- style = *linear* or *wiggle* or *rotate* or *variable*

```
  linear args = Vx Vy Vz
      Vx,Vy,Vz = components of velocity vector (velocity units), any component can be specified
    wiggle args = Ax Ay Az period
      Ax,Ay,Az = components of amplitude vector (distance units), any component can be specifie
      period = period of oscillation (time units)
    rotate args = Px Py Pz Rx Ry Rz period
      Px,Py,Pz = origin point of axis of rotation (distance units)
      Rx,Ry,Rz = axis of rotation vector
      period = period of rotation (time units)
    variable args = v_dx v_dy v_dz v_vx v_vy v_vz
      v_dx,v_dy,v_dz = 3 variable names that calculate x,y,z displacement as function of time, a
      v_vx,v_vy,v_vz = 3 variable names that calculate x,y,z velocity as function of time, any c
```

- zero or more keyword/value pairs may be appended
- keyword = *units*

```
  units value = box or lattice
```

Examples:

```
fix 1 boundary move wiggle 3.0 0.0 0.0 1.0 units box
fix 2 boundary move rotate 0.0 0.0 0.0 0.0 0.0 1.0 5.0
fix 2 boundary move variable v_myx v_myy NULL v_VX v_VY NULL
```

Description:

Perform updates of position and velocity for atoms in the group each timestep using the specified settings or formulas, without regard to forces on the atoms. This can be useful for boundary or other atoms, whose movement can influence nearby atoms.

IMPORTANT NOTE: The atoms affected by this fix should not normally be time integrated by other fixes (e.g. fix nve, fix nvt), since that will change their positions and velocities twice.

IMPORTANT NOTE: As atoms move due to this fix, they will pass thru periodic boundaries and be remapped to the other side of the simulation box, just as they would during normal time integration (e.g. via the fix nve command). It is up to you to decide whether periodic boundaries are appropriate with the kind of atom motion you are prescribing with this fix.

IMPORTANT NOTE: As dicsussed below, atoms are moved relative to their initial position at the time the fix is specified. These initial coordinates are stored by the fix in "unwrapped" form, by using the image flags associated with each atom. See the dump custom command for a discussion of "unwrapped" coordinates. See the Atoms section of the read_data command for a discussion of image flags and how they are set for each atom. You can reset the image flags (e.g. to 0) before invoking this fix by using the set image command.

The *linear* style moves atoms at a constant velocity, so that their position X = (x,y,z) as a function of time is given in vector notation as

```
X(t) = X0 + V * delta
```

where *X0* = (x0,y0,z0) is their position at the time the fix is specified, *V* is the specified velocity vector with components (Vx,Vy,Vz), and *delta* is the time elapsed since the fix was specified. This style also sets the velocity of each atom to V = (Vx,Vy,Vz). If any of the velocity components is specified as NULL, then the position and velocity of that component is time integrated the same as the fix nve command would perform, using the corresponding force component on the atom.

The *wiggle* style moves atoms in an oscillatory fashion, so that their position X = (x,y,z) as a function of time is given in vector notation as

```
X(t) = X0 + A sin(omega*delta)
```

where *X0* = (x0,y0,z0) is their position at the time the fix is specified, *A* is the specified amplitude vector with components (Ax,Ay,Az), *omega* is 2 PI / *period*, and *delta* is the time elapsed since the fix was specified. This style also sets the velocity of each atom to the time derivative of this expression. If any of the amplitude components is specified as NULL, then the position and velocity of that component is time integrated the same as the fix nve command would perform, using the corresponding force component on the atom.

The *rotate* style rotates atoms around a rotation axis R = (Rx,Ry,Rz) that goes thru a point P = (Px,Py,Pz). The *period* of the rotation is also specified. This style also sets the velocity of each atom to (omega cross Rperp) where omega is its angular velocity around the rotation axis and Rperp is a perpendicular vector from the rotation axis to the atom. If the defined atom_style assigns an angular velocity to each atom, then each atom's angular velocity is also set to omega. Note that the direction of rotation for the atoms around the rotation axis is consistent with the right−hand rule: if your right−hand's thumb points along *R*, then your fingers wrap around the axis in the direction of rotation.

The *variable* style allows the position and velocity components of each atom to be set by formulas specified via the variable command. Each of the 6 variables is specified as an argument to the fix as v_name, where name is the variable name that is defined elsewhere in the input script.

Each variable must be of either the *equal* or *atom* style. *Equal*−style variables compute a single numeric quantity, that can be a function of the timestep as well as of other simulation values. *Atom*−style variables compute a numeric quantity for each atom, that can be a function per−atom quantities, such as the atom's position, as well as of the timestep and other simulation values. Note that this fix stores the original coordinates of each atom (see note below) so that per−atom quantity can be used in an atom−style variable formula. See the variable command for details.

The first 3 variables (v_dx,v_dy,v_dz) specified for the *variable* style are used to calculate a displacement from the atom's original position at the time the fix was specified. The second 3 variables (v_vx,v_vy,v_vz) specified are used to compute a velocity for each atom.

Any of the 6 variables can be specified as NULL. If both the displacement and velocity variables for a particular x,y,z component are specified as NULL, then the position and velocity of that component is time integrated the same as the fix nve command would perform, using the corresponding force component on the atom. If only the velocity variable for a component is specified as NULL, then the displacement variable will be used to set the position of the atom, and its velocity component will not be changed. If only the displacement variable for a component is specified as NULL, then the velocity variable will be used to set the velocity of the atom, and the position of the atom will be time integrated using that velocity.

The *units* keyword determines the meaning of the distance units used to define the *linear* velocity and *wiggle* amplitude and *rotate* origin. This setting is ignored for the *variable* style. A *box* value selects standard units as defined by the units command, e.g. velocity in Angstroms/fmsec and amplitude and position in Angstroms for units = real. A *lattice* value means the velocity units are in lattice spacings per time and the amplitude and position are in lattice spacings. The lattice command must have been previously used to define the lattice spacing. Each of these 3 quantities may be dependent on the x,y,z dimension, since the lattice spacings can be different in x,y,z.

For rRESPA time integration, this fix adjusts the position and velocity of atoms on the outermost rRESPA level.

Restart, fix_modify, output, run start/stop, minimize info:

This fix writes the original coordinates of moving atoms to binary restart files, so that the motion can be continuous in a restarted simulation. See the read_restart command for info on how to re−specify a fix in an input script that reads a restart file, so that the operation of the fix continues in an uninterrupted fashion.

None of the fix_modify options are relevant to this fix.

This fix produces a per−atom array which can be accessed by various output commands. The number of columns for each atom is 3, and the columns store the original unwrapped x,y,z coords of each atom. The per−atom values can be accessed on any timestep.

No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions: none

Related commands:

fix nve

Default: none

The option default is units = lattice.

fix msst command

Syntax:

```
fix ID group-ID msst dir shockvel keyword value ...
```

- ID, group−ID are documented in fix command
- msst = style name of this fix
- dir = *x* or *y* or *z*
- shockvel = shock velocity (strictly positive, distance/time units)
- zero or more keyword value pairs may be appended
- keyword = *q* or *mu* or *p0* or *v0* or *e0* or *tscale*

```
  q value = cell mass-like parameter (mass^2/distance^4 units)
    mu value = artificial viscosity (mass/length/time units)
    p0 value = initial pressure in the shock equations (pressure units)
    v0 value = initial simulation cell volume in the shock equations (distance^3 units)
    e0 value = initial total energy (energy units)
    tscale value = reduction in initial temperature (unitless fraction between 0.0 and 1.0)
```

Examples:

```
fix 1 all msst y 100.0 q 1.0e5 mu 1.0e5
fix 2 all msst z 50.0 q 1.0e4 mu 1.0e4  v0 4.3419e+03 p0 3.7797e+03 e0 -9.72360e+02 tscale 0.01
```

Description:

This command performs the Multi−Scale Shock Technique (MSST) integration to update positions and velocities each timestep to mimic a compressive shock wave passing over the system. See (Reed) for a detailed description of this method. The MSST varies the cell volume and temperature in such a way as to restrain the system to the shock Hugoniot and the Rayleigh line. These restraints correspond to the macroscopic conservation laws dictated by a shock front. *shockvel* determines the steady shock velocity that will be simulated.

To perform a simulation, choose a value of *q* that provides volume compression on the timescale of 100 fs to 1 ps. If the volume is not compressing, either the shock speed is chosen to be below the material sound speed or *p0* has been chosen inaccurately. Volume compression at the start can be sped up by using a non−zero value of *tscale*. Use the smallest value of *tscale* that results in compression.

Under some special high−symmetry conditions, the pressure (volume) and/or temperature of the system may oscillate for many cycles even with an appropriate choice of mass−like parameter *q*. Such oscillations have physical significance in some cases. The optional *mu* keyword adds an artificial viscosity that helps break the system symmetry to equilibrate to the shock Hugoniot and Rayleigh line more rapidly in such cases.

tscale is a factor between 0 and 1 that determines what fraction of thermal kinetic energy is converted to compressive strain kinetic energy at the start of the simulation. Setting this parameter to a non−zero value may assist in compression at the start of simulations where it is slow to occur.

If keywords *e0*, *p0*,or *v0* are not supplied, these quantities will be calculated on the first step, after the energy specified by *tscale* is removed. The value of *e0* is not used in the dynamical equations, but is used in calculating the deviation from the Hugoniot.

Values of shockvel less than a critical value determined by the material response will not have compressive solutions. This will be reflected in lack of significant change of the volume in the MSST.

For all pressure styles, the simulation box stays orthogonal in shape. Parrinello−Rahman boundary conditions (tilted box) are supported by LAMMPS, but are not implemented for MSST.

This fix computes a temperature and pressure each timestep. To do this, the fix creates its own computes of style "temp" and "pressure", as if these commands had been issued:

```
compute fix-ID_temp group-ID temp
compute fix-ID_press group-ID pressure fix-ID_temp
```

See the compute temp and compute pressure commands for details. Note that the IDs of the new computes are the fix−ID + underscore + "temp" or fix_ID + underscore + "press". The group for the new computes is "all".

Restart, fix_modify, output, run start/stop, minimize info:

This fix writes the state of all internal variables to binary restart files. See the read_restart command for info on how to re−specify a fix in an input script that reads a restart file, so that the operation of the fix continues in an uninterrupted fashion.

The progress of the MSST can be monitored by printing the global scalar and global vector quantities computed by the fix.

The scalar is the cumulative energy change due to the fix. This is also the energy added to the potential energy by the fix_modify *energy* command. With this command, the thermo keyword *etotal* prints the conserved quantity of the MSST dynamic equations. This can be used to test if the MD timestep is sufficiently small for accurate integration of the dynamic equations. See also thermo_style command.

The global vector contains four values in this order:

[*dhugoniot*, *drayleigh*, *lagrangian_speed*, *lagrangian_position*]

1. *dhugoniot* is the departure from the Hugoniot (temperature units).
2. *drayleigh* is the departure from the Rayleigh line (pressure units).
3. *lagrangian_speed* is the laboratory−frame Lagrangian speed (particle velocity) of the computational cell (velocity units).
4. *lagrangian_position* is the computational cell position in the reference frame moving at the shock speed. This is usually a good estimate of distance of the computational cell behind the shock front.

To print these quantities to the log file with descriptive column headers, the following LAMMPS commands are suggested:

```
fix             msst all msst z
fix_modify      msst energy yes
variable dhug   equal f_msst[1]
variable dray   equal f_msst[2]
variable lgr_vel equal f_msst[3]
variable lgr_pos equal f_msst[4]
thermo_style    custom step temp ke pe lz pzz etotal v_dhug v_dray v_lgr_vel v_lgr_pos f_msst
```

These fixes compute a global scalar and a global vector of 4 quantities, which can be accessed by various output commands. The scalar values calculated by this fix are "extensive"; the vector values are "intensive".

Restrictions:

This fix style is part of the "shock" package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

All cell dimensions must be periodic. This fix can not be used with a triclinic cell. The MSST fix has been tested only for the group−ID all.

Related commands:

fix deform

Default:

The keyword defaults are q = 10, mu = 0, tscale = 0.01. p0, v0, and e0 are calculated on the first step.

(Reed) Reed, Fried, and Joannopoulos, Phys. Rev. Lett., 90, 235503 (2003).

fix neb command

Syntax:

```
fix ID group−ID neb Kspring
```

- ID, group−ID are documented in fix command
- neb = style name of this fix command
- Kspring = inter−replica spring constant (force/distance units)

Examples:

```
fix 1 active neb 10.0
```

Description:

Add inter−replica forces to atoms in the group for a multi−replica simulation run via the neb command to perform a nudged elastic band (NEB) calculation for transition state finding. Hi−level explanations of NEB are given with the neb command and in this section of the manual. The fix neb command must be used with the "neb" command to define how inter−replica forces are computed.

Only the N atoms in the fix group experience inter−replica forces. Atoms in the two end−point replicas do not experience these forces, but those in intermediate replicas do. During the initial stage of NEB, the 3N−length vector of interatomic forces Fi = −Grad(V) acting on the atoms of each intermediate replica I is altered, as described in the (Henkelman1) paper, to become:

```
Fi = −Grad(V) + (Grad(V) dot That) That + Kspring (|Ri+i − Ri| − |Ri − Ri−1|) That
```

Ri are the atomic coordinates of replica I; Ri−1 and Ri+1 are the coordinates of its neighbor replicas. That (t with a hat over it) is the unit "tangent" vector for replica I which is a function of Ri, Ri−1, Ri+1, and the potential energy of the 3 replicas; it points roughly in the direction of (Ri+i − Ri−1); see the (Henkelman1) paper for details.

The first two terms in the above equation are the component of the interatomic forces perpendicular to the tangent vector. The last term is a spring force between replica I and its neighbors, parallel to the tangent vector direction with the specified spring constant *Kspring*.

The effect of the first two terms is to push the atoms of each replica toward the minimum energy path (MEP) of conformational states that transition over the energy barrier. The MEP for an energy barrier is defined as a sequence of 3N−dimensional states which cross the barrier at its saddle point, each of which has a potential energy gradient parallel to the MEP itself.

The effect of the last term is to push each replica away from its two neighbors in a direction along the MEP, so that the final set of states are equidistant from each other.

During the second stage of NEB, the forces on the N atoms in the replica nearest the top of the energy barrier are altered so that it climbs to the top of the barrier and finds the saddle point. The forces on atoms in this replica are described in the (Henkelman2) paper, and become:

```
Fi = −Grad(V) + 2 (Grad(V) dot That) That
```

The inter−replica forces for the other replicas are unchanged from the first equation.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix. No global or per−atom quantities are stored by this fix for access by various output commands. No parameter of this fix can be used with the *start/stop* keywords of the run command.

The forces due to this fix are imposed during an energy minimization, as invoked by the minimize command via the neb command.

Restrictions:

This command can only be used if LAMMPS was built with the "replica" package. See the Making LAMMPS section for more info on packages.

Related commands:

neb

Default: none

(Henkelman1) Henkelman and Jonsson, J Chem Phys, 113, 9978−9985 (2000).

(Henkelman2) Henkelman, Uberuaga, Jonsson, J Chem Phys, 113, 9901−9904 (2000).

fix nvt command

fix nvt/cuda command

fix npt command

fix npt/cuda command

fix nph command

Syntax:

```
fix ID group-ID style_name keyword value ...
```

- ID, group−ID are documented in fix command
- style_name = *nvt* or *npt* or *nph*

```
one or more keyword value pairs may be appended
keyword = temp or iso or aniso or tri or x or y or z or xy or yz or xz or couple or tchain or
  temp values = Tstart Tstop Tdamp
    Tstart,Tstop = external temperature at start/end of run
    Tdamp = temperature damping parameter (time units)
  iso or aniso or tri values = Pstart Pstop Pdamp
    Pstart,Pstop = scalar external pressure at start/end of run (pressure units)
    Pdamp = pressure damping parameter (time units)
  x or y or z or xy or yz or xz values = Pstart Pstop Pdamp
    Pstart,Pstop = external stress tensor component at start/end of run (pressure units)
    Pdamp = stress damping parameter (time units)
  couple = none or xyz or xy or yz or xz
  tchain value = length of thermostat chain (1 = single thermostat)
  pchain values = length of thermostat chain on barostat (0 = no thermostat)
  mtk value = yes or no = add in MTK adjustment term or not
  tloop value = number of sub-cycles to perform on thermostat
  ploop value = number of sub-cycles to perform on barostat thermostat
  nreset value = reset reference cell every this many timesteps
  drag value = drag factor added to barostat/thermostat (0.0 = no drag)
  dilate value = all or partial
  scaleyz value = yes or no = scale yz with lz
  scalexz value = yes or no = scale xz with lz
  scalexy value = yes or no = scale xy with ly
```

Examples:

```
fix 1 all nvt temp 300.0 300.0 100.0
fix 1 water npt temp 300.0 300.0 100.0 iso 0.0 0.0 1000.0
fix 2 jello npt temp 300.0 300.0 100.0 tri 5.0 5.0 1000.0
fix 2 ice nph x 1.0 1.0 0.5 y 2.0 2.0 0.5 z 3.0 3.0 0.5 yz 0.1 0.1 0.5 xz 0.2 0.2 0.5 xy 0.3 0.3 0.5
```

Description:

These commands perform time integration on Nose−Hoover style non−Hamiltonian equations of motion which are designed to generate positions and velocities sampled from the canonical (nvt), isothermal−isobaric (npt), and isenthalpic (nph) ensembles. This is achieved by adding some dynamic variables which are coupled to the particle velocities (thermostatting) and simulation domain dimensions (barostatting). In addition to basic thermostatting

and barostatting, these fixes can also create a chain of thermostats coupled to the particle thermostat, and another chain of thermostats coupled to the barostat variables. The barostat can be coupled to the overall box volume, or to individual dimensions, including the *xy*, *xz* and *yz* tilt dimensions. The external pressure of the barostat can be specified as either a scalar pressure (isobaric ensemble) or as components of a symmetric stress tensor (constant stress ensemble). When used correctly, the time−averaged temperature and stress tensor of the particles will match the target values specified by Tstart/Tstop and Pstart/Pstop.

The equations of motion used are those of Shinoda et al. in (Shinoda), which combine the hydrostatic equations of Martyna, Tobias and Klein in (Martyna) with the strain energy proposed by Parrinello and Rahman in (Parrinello). The time integration schemes closely follow the time−reversible measure−preserving Verlet and rRESPA integrators derived by Tuckerman et al. in (Tuckerman).

The thermostat for fix styles *nvt* and *npt* is specified using the *temp* keyword. Other thermostat−related keywords are *tchain*, *tloop* and *drag*, which are discussed below.

The thermostat is applied to only the translational degrees of freedom for the particles. The translational degrees of freedom can also have a bias velocity removed before thermostatting takes place; see the description below. The desired temperature at each timestep is a ramped value during the run from *Tstart* to *Tstop*. The *Tdamp* parameter is specified in time units and determines how rapidly the temperature is relaxed. For example, a value of 10.0 means to relax the temperature in a timespan of (roughly) 10 time units (e.g. tau or fmsec or psec − see the units command). The atoms in the fix group are the only ones whose velocities and positions are updated by the velocity/position update portion of the integration.

IMPORTANT NOTE: A Nose−Hoover thermostat will not work well for arbitrary values of *Tdamp*. If *Tdamp* is too small, the temperature can fluctuate wildly; if it is too large, the temperature will take a very long time to equilibrate. A good choice for many models is a *Tdamp* of around 100 timesteps. Note that this is NOT the same as 100 time units for most units settings.

The barostat for fix styles *npt* and *nph* is specified using one or more of the *iso*, *aniso*, *tri*, *x*, *y*, *z*, *xy*, *xz*, *yz*, and *couple* keywords. These keywords give you the ability to specify all 6 components of an external stress tensor, and to couple various of these components together so that the dimensions they represent are varied together during a constant−pressure simulation.

Other barostat−related keywords are *pchain*, *mtk*, *ploop*, *nreset*, *drag*, and *dilate*, which are discussed below.

Orthogonal simulation boxes have 3 adjustable dimensions (x,y,z). Triclinic (non−orthogonal) simulation boxes have 6 adjustable dimensions (x,y,z,xy,xz,yz). The create_box, read data, and read_restart commands specify whether the simulation box is orthogonal or non−orthogonal (triclinic) and explain the meaning of the xy,xz,yz tilt factors.

The target pressures for each of the 6 components of the stress tensor can be specified independently via the *x*, *y*, *z*, *xy*, *xz*, *yz* keywords, which correspond to the 6 simulation box dimensions. For each component, the external pressure or tensor component at each timestep is a ramped value during the run from *Pstart* to *Pstop*. If a target pressure is specified for a component, then the corresponding box dimension will change during a simulation. For example, if the *y* keyword is used, the y−box length will change. If the *xy* keyword is used, the xy tilt factor will change. A box dimension will not change if that component is not specified, although you have the option to change that dimension via the fix deform command.

Note that in order to use the *xy*, *xz*, or *yz* keywords, the simulation box must be triclinic, even if its initial tilt factors are 0.0.

For all barostat keywords, the *Pdamp* parameter operates like the *Tdamp* parameter, determining the time scale on which pressure is relaxed. For example, a value of 10.0 means to relax the pressure in a timespan of (roughly) 10 time units (e.g. tau or fmsec or psec − see the units command).

IMPORTANT NOTE: A Nose−Hoover barostat will not work well for arbitrary values of *Pdamp*. If *Pdamp* is too small, the pressure and volume can fluctuate wildly; if it is too large, the pressure will take a very long time to equilibrate. A good choice for many models is a *Pdamp* of around 1000 timesteps. Note that this is NOT the same as 1000 time units for most units settings.

Regardless of what atoms are in the fix group, a global pressure or stress tensor is computed for all atoms. Similarly, when the size of the simulation box is changed, all atoms are re−scaled to new positions, unless the keyword *dilate* is specified with a value of *partial*, in which case only the atoms in the fix group are re−scaled. The latter can be useful for leaving the coordinates of atoms in a solid substrate unchanged and controlling the pressure of a surrounding fluid.

The *couple* keyword allows two or three of the diagonal components of the pressure tensor to be "coupled" together. The value specified with the keyword determines which are coupled. For example, *xz* means the *Pxx* and *Pzz* components of the stress tensor are coupled. *Xyz* means all 3 diagonal components are coupled. Coupling means two things: the instantaneous stress will be computed as an average of the corresponding diagonal components, and the coupled box dimensions will be changed together in lockstep, meaning coupled dimensions will be dilated or contracted by the same percentage every timestep. The *Pstart*, *Pstop*, *Pdamp* parameters for any coupled dimensions must be identical. *Couple xyz* can be used for a 2d simulation; the *z* dimension is simply ignored.

The *iso*, *aniso*, and *tri* keywords are simply shortcuts that are equivalent to specifying several other keywords together.

The keyword *iso* means couple all 3 diagonal components together when pressure is computed (hydrostatic pressure), and dilate/contract the dimensions together. Using "iso Pstart Pstop Pdamp" is the same as specifying these 4 keywords:

```
x Pstart Pstop Pdamp
y Pstart Pstop Pdamp
z Pstart Pstop Pdamp
couple xyz
```

The keyword *aniso* means *x*, *y*, and *z* dimensions are controlled independently using the *Pxx*, *Pyy*, and *Pzz* components of the stress tensor as the driving forces, and the specified scalar external pressure. Using "aniso Pstart Pstop Pdamp" is the same as specifying these 4 keywords:

```
x Pstart Pstop Pdamp
y Pstart Pstop Pdamp
z Pstart Pstop Pdamp
couple none
```

The keyword *tri* means *x*, *y*, *z*, *xy*, *xz*, and *yz* dimensions are controlled independently using their individual stress components as the driving forces, and the specified scalar pressure as the external normal stress. Using "tri Pstart Pstop Pdamp" is the same as specifying these 7 keywords:

```
x Pstart Pstop Pdamp
y Pstart Pstop Pdamp
z Pstart Pstop Pdamp
xy 0.0 0.0 Pdamp
yz 0.0 0.0 Pdamp
xz 0.0 0.0 Pdamp
```

In some cases (e.g. for solids) the pressure (volume) and/or temperature of the system can oscillate undesirably when a Nose/Hoover barostat and thermostat is applied. The optional *drag* keyword will damp these oscillations, although it alters the Nose/Hoover equations. A value of 0.0 (no drag) leaves the Nose/Hoover formalism unchanged. A non−zero value adds a drag term; the larger the value specified, the greater the damping effect. Performing a short run and monitoring the pressure and temperature is the best way to determine if the drag term is working. Typically a value between 0.2 to 2.0 is sufficient to damp oscillations after a few periods. Note that use of the drag keyword will interfere with energy conservation and will also change the distribution of positions and velocities so that they do not correspond to the nominal NVT, NPT, or NPH ensembles.

An alternative way to control initial oscillations is to use chain thermostats. The keyword *tchain* determines the number of thermostats in the particle thermostat. A value of 1 corresponds to the original Nose−Hoover thermostat. The keyword *pchain* specifies the number of thermostats in the chain thermostatting the barostat degrees of freedom. A value of 0 corresponds to no thermostatting of the barostat variables.

The *mtk* keyword controls whether or not the correction terms due to Martyna, Tuckerman, and Klein are included in the equations of motion (Martyna). Specifying *no* reproduces the original Hoover barostat, whose volume probability distribution function differs from the true NPT and NPH ensembles by a factor of 1/V. Hence using *yes* is more correct, but in many cases the difference is negligible.

The keyword *tloop* can be used to improve the accuracy of integration scheme at little extra cost. The initial and final updates of the thermostat variables are broken up into *tloop* substeps, each of length *dt*/*tloop*. This corresponds to using a first−order Suzuki−Yoshida scheme (Tuckerman2006). The keyword *ploop* does the same thing for the barostat thermostat.

The keyword *nreset* controls how often the reference dimensions used to define the strain energy are reset. If this keyword is not used, or is given a value of zero, then the reference dimensions are set to those of the initial simulation domain and are never changed. If the simulation domain changes significantly during the simulation, then the final average pressure tensor will differ significantly from the specified values of the external stress tensor. A value of *nstep* means that every *nstep* timesteps, the reference dimensions are set to those of the current simulation domain.

The *scaleyz*, *scalexz*, and *scalexy* keywords control whether or not the corresponding tilt factors are scaled with the associated box dimensions when barostatting triclinic periodic cells. The default values *yes* will turn on scaling, which corresponds to adjusting the linear dimensions of the cell while preserving its shape. Choosing *no* ensures that the tilt factors are not scaled with the box dimensions. See below for restrictions and default values in different situations. In older versions of LAMMPS, scaling of tilt factors was not performed. The old behavior can be recovered by setting all three scale keywords to *no*.

IMPORTANT NOTE: Using a barostat coupled to tilt dimensions *xy*, *xz*, *yz* can sometimes result in arbitrarily large values of the tilt dimensions, i.e. a dramatically deformed simulation box. LAMMPS allows the tilt factors to grow a little beyond the normal limit of half the box length (0.6 times the box length), and then performs flipping or re−shaping to an equivalent periodic cell. The re−shaping operation is described in more detail in the doc page for fix deform. Both the barostat dynamics and the atom trajectories are unaffected by this operation. However, if a tilt factor is incremented by a large amount (1.5 times the box length) on a single timestep, LAMMPS can not accomodate this event and will terminate the simulation with an error. This error typically indicates that there is something badly wrong with how the simulation was constructed, such as specifying values of *Pstart* that are too far from the current stress value, or specifying a timestep that is too large. Triclinic barostatting should be used with care. This also is true for other barostat styles, although they tend to be more forgiving of insults. In particular, it is important to recognize that equilibrium liquids can not support a shear stress and that equilibrium solids can not support shear stresses that exceed the yield stress.

IMPORTANT NOTE: Unlike the fix temp/berendsen command which performs thermostatting but NO time integration, these fixes perform thermostatting/barostatting AND time integration. Thus you should not use any other time integration fix, such as fix nve on atoms to which this fix is applied. Likewise, the *temp* options for fix nvt and fix npt should not normally be used on atoms that also have their temperature controlled by another fix − e.g. by fix langevin or fix temp/rescale commands.

See this howto section of the manual for a discussion of different ways to compute temperature and perform thermostatting and barostatting.

These fixes compute a temperature and pressure each timestep. To do this, the fix creates its own computes of style "temp" and "pressure", as if one of these two sets of commands had been issued:

```
compute fix-ID_temp group-ID temp
compute fix-ID_press group-ID pressure fix-ID_temp

compute fix-ID_temp all temp
compute fix-ID_press all pressure fix-ID_temp
```

See the compute temp and compute pressure commands for details. Note that the IDs of the new computes are the fix−ID + underscore + "temp" or fix_ID + underscore + "press". For fix nvt, the group for the new computes is the same as the fix group. For fix nph and fix npt, the group for the new computes is "all" since pressure is computed for the entire system.

Note that these are NOT the computes used by thermodynamic output (see the thermo_style command) with ID = *thermo_temp* and *thermo_press*. This means you can change the attributes of this fix's temperature or pressure via the compute_modify command or print this temperature or pressure during thermodynamic output via the thermo_style custom command using the appropriate compute−ID. It also means that changing attributes of *thermo_temp* or *thermo_press* will have no effect on this fix.

Like other fixes that perform thermostatting, fix nvt and fix npt can be used with compute commands that calculate a temperature after removing a "bias" from the atom velocities. E.g. removing the center−of−mass velocity from a group of atoms or only calculating temperature on the x−component of velocity or only calculating temperature for atoms in a geometric region. This is not done by default, but only if the fix_modify command is used to assign a temperature compute to this fix that includes such a bias term. See the doc pages for individual compute commands to determine which ones include a bias. In this case, the thermostat works in the following manner: the current temperature is calculated taking the bias into account, bias is removed from each atom, thermostatting is performed on the remaining thermal degrees of freedom, and the bias is added back in.

Styles with a *cuda* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in this section of the manual. The accelerated styles take the same arguments and should produce the same results, except for round−off and precision issues.

These accelerated styles are part of the "user−cuda" package. They are only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the −suffix command−line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See this section of the manual for more instructions on how to use the accelerated styles effectively.

Restart, fix_modify, output, run start/stop, minimize info:

These fixes writes the state of all the thermostat and barostat variables to binary restart files. See the read_restart command for info on how to re−specify a fix in an input script that reads a restart file, so that the operation of the fix continues in an uninterrupted fashion.

The fix_modify *temp* and *press* options are supported by these fixes. You can use them to assign a compute you have defined to this fix which will be used in its thermostatting or barostatting procedure, as described above. If you do this, note that the kinetic energy derived from the compute temperature should be consistent with the virial term computed using all atoms for the pressure. LAMMPS will warn you if you choose to compute temperature on a subset of atoms.

IMPORTANT NOTE: If both the *temp* and *press* keywords are used in a single thermo_modify command (or in two separate commands), then the order in which the keywords are specified is important. Note that a pressure compute defines its own temperature compute as an argument when it is specified. The *temp* keyword will override this (for the pressure compute being used by fix npt), but only if the *temp* keyword comes after the *press* keyword. If the *temp* keyword comes before the *press* keyword, then the new pressure compute specified by the *press* keyword will be unaffected by the *temp* setting.

The fix_modify *energy* option is supported by these fixes to add the energy change induced by Nose/Hoover thermostatting and barostatting to the system's potential energy as part of thermodynamic output.

These fixes compute a global scalar and a global vector of quantities, which can be accessed by various output commands. The scalar value calculated by these fixes is "extensive"; the vector values are "intensive".

The scalar is the cumulative energy change due to the fix.

The vector stores internal Nose/Hoover thermostat and barostat variables. The number and meaning of the vector values depends on which fix is used and the settings for keywords *tchain* and *pchain*, which specify the number of Nose/Hoover chains for the thermostat and barostat. If no thermostatting is done, then *tchain* is 0. If no barostatting is done, then *pchain* is 0. In the following list, "ndof" is 0, 1, 3, or 6, and is the number of degrees of freedom in the barostat. Its value is 0 if no barostat is used, else its value is 6 if any off−diagonal stress tensor component is barostatted, else its value is 1 if *couple xyz* is used or *couple xy* for a 2d simulation, otherwise its value is 3.

The order of values in the global vector and their meaning is as follows. The notation means there are tchain values for eta, followed by tchain for eta_dot, followed by ndof for omega, etc:

- eta[tchain] = particle thermostat displacements (unitless)
- eta_dot[tchain] = particle thermostat velocities (1/time units)
- omega[ndof] = barostat displacements (unitless)
- omega_dot[ndof] = barostat velocities (1/time units)
- etap[pchain] = barostat thermostat displacements (unitless)
- etap_dot[pchain] = barostat thermostat velocities (1/time units)
- PE_eta[tchain] = potential energy of each particle thermostat displacement (energy units)
- KE_eta_dot[tchain] = kinetic energy of each particle thermostat velocity (energy units)
- PE_omega[ndof] = potential energy of each barostat displacement (energy units)
- KE_omega_dot[ndof] = kinetic energy of each barostat velocity (energy units)
- PE_etap[pchain] = potential energy of each barostat thermostat displacement (energy units)
- KE_etap_dot[pchain] = kinetic energy of each barostat thermostat velocity (energy units)
- PE_strain[1] = scalar strain energy (energy units)

These fixes can ramp their external temperature and pressure over multiple runs, using the *start* and *stop* keywords of the run command. See the run command for details of how to do this.

These fixes are not invoked during energy minimization.

These fixes can be used with either the *verlet* or *respa* integrators. When using one of the barostat fixes with *respa*, LAMMPS uses an integrator constructed according to the following factorization of the Liouville propagator (for two rRESPA levels):

$$
\begin{aligned}
\Delta t) \quad = \quad & \exp\left(\mathrm{i}L_{\text{T-baro}}\frac{\Delta t}{2}\right)\exp\left(\mathrm{i}L_{\text{T-part}}\frac{\Delta t}{2}\right)\exp\left(\mathrm{i}L_{\epsilon,2}\frac{\Delta t}{2}\right)\exp\left(\mathrm{i} \right. \\
& \times \left[\exp\left(\mathrm{i}L_2^{(1)}\frac{\Delta t}{2n}\right)\exp\left(\mathrm{i}L_{\epsilon,1}\frac{\Delta t}{n}\right)\exp\left(\mathrm{i}L_1\frac{\Delta t}{n}\right)\exp\left(\mathrm{i}L_2^{(1)}\right.\right. \\
& \times \exp\left(\mathrm{i}L_2^{(2)}\frac{\Delta t}{2}\right)\exp\left(\mathrm{i}L_{\epsilon,2}\frac{\Delta t}{2}\right)\exp\left(\mathrm{i}L_{\text{T-part}}\frac{\Delta t}{2}\right)\exp\left(\mathrm{i}L\right. \\
& +\mathcal{O}\left(\Delta t^3\right)
\end{aligned}
$$

This factorization differs somewhat from that of Tuckerman et al., in that the barostat is only updated at the outermost rRESPA level, whereas Tuckerman's factorization requires splitting the pressure into pieces corresponding to the forces computed at each rRESPA level. In theory, the latter method will exhibit better numerical stability. In practice, because Pdamp is normally chosen to be a large multiple of the outermost rRESPA timestep, the barostat dynamics are not the limiting factor for numerical stability. Both factorizations are time−reversible and can be shown to preserve the phase space measure of the underlying non−Hamiltonian equations of motion.

Restrictions:

Non−periodic dimensions cannot be barostatted. *Z*, *xz*, and *yz*, cannot be barostatted 2D simulations. *Xy*, *xz*, and *yz* can only be barostatted if the simulation domain is triclinic and the 2nd dimension in the keyword (*y* dimension in *xy*) is periodic. The create_box, read data, and read_restart commands specify whether the simulation box is orthogonal or non−orthogonal (triclinic) and explain the meaning of the xy,xz,yz tilt factors.

For the *temp* keyword, the final Tstop cannot be 0.0 since it would make the external T = 0.0 at some timestep during the simulation which is not allowed in the Nose/Hoover formulation.

The *scaleyz yes* and *scalexz yes* keyword/value pairs can not be used for 2D simulations. *scaleyz yes*, *scalexz yes*, and *scalexy yes* options can only be used if the 2nd dimension in the keyword is periodic, and if the tilt factor is not coupled to the barostat via keywords *tri*, *yz*, *xz*, and *xy*.

Related commands:

fix nve, fix_modify, run_style

Default:

The keyword defaults are tchain = 3, pchain = 3, mtk = yes, tloop = ploop = 1, nreset = 0, drag = 0.0, dilate = all, couple = none, scaleyz = scalexz = scalexy = yes if periodic in 2nd dimension and not coupled to barostat, otherwise no.

(Martyna) Martyna, Tobias and Klein, J Chem Phys, 101, 4177 (1994).

(Parrinello) Parrinello and Rahman, J Appl Phys, 52, 7182 (1981).

(Tuckerman) Tuckerman, Alejandre, Lopez−Rendon, Jochim, and Martyna, J Phys A: Math Gen, 39, 5629 (2006).

(Shinoda) Shinoda, Shiga, and Mikami, Phys Rev B, 69, 134103 (2004).

fix nvt/eff command

fix npt/eff command

fix nph/eff command

Syntax:

```
fix ID group−ID style_name keyword value ...
```

- ID, group−ID are documented in fix command
- style_name = *nvt/eff* or *npt/eff* or *nph/eff*

```
one or more keyword value pairs may be appended
keyword = temp or iso or aniso or tri or x or y or z or xy or yz or xz or couple or tchain or
  temp values = Tstart Tstop Tdamp
    Tstart,Tstop = external temperature at start/end of run
    Tdamp = temperature damping parameter (time units)
  iso or aniso or tri values = Pstart Pstop Pdamp
    Pstart,Pstop = scalar external pressure at start/end of run (pressure units)
    Pdamp = pressure damping parameter (time units)
  x or y or z or xy or yz or xz values = Pstart Pstop Pdamp
    Pstart,Pstop = external stress tensor component at start/end of run (pressure units)
    Pdamp = stress damping parameter (time units)
  couple = none or xyz or xy or yz or xz
  tchain value = length of thermostat chain (1 = single thermostat)
  pchain values = length of thermostat chain on barostat (0 = no thermostat)
  mtk value = yes or no = add in MTK adjustment term or not
  tloop value = number of sub−cycles to perform on thermostat
  ploop value = number of sub−cycles to perform on barostat thermostat
  nreset value = reset reference cell every this many timesteps
  drag value = drag factor added to barostat/thermostat (0.0 = no drag)
  dilate value = all or partial
```

Examples:

```
fix 1 all nvt/eff temp 300.0 300.0 0.1
fix 1 part npt/eff temp 300.0 300.0 0.1 iso 0.0 0.0 1.0
fix 2 part npt/eff temp 300.0 300.0 0.1 tri 5.0 5.0 1.0
fix 2 ice nph/eff x 1.0 1.0 0.5 y 2.0 2.0 0.5 z 3.0 3.0 0.5 yz 0.1 0.1 0.5 xz 0.2 0.2 0.5 xy 0.3 0.3
```

Description:

These commands perform time integration on Nose−Hoover style non−Hamiltonian equations of motion for nuclei and electrons in the group for the electron force field model. The fixes are designed to generate positions and velocities sampled from the canonical (nvt), isothermal−isobaric (npt), and isenthalpic (nph) ensembles. This is achieved by adding some dynamic variables which are coupled to the particle velocities (thermostatting) and simulation domain dimensions (barostatting). In addition to basic thermostatting and barostatting, these fixes can also create a chain of thermostats coupled to the particle thermostat, and another chain of thermostats coupled to the barostat variables. The barostat can be coupled to the overall box volume, or to individual dimensions, including the *xy*, *xz* and *yz* tilt dimensions. The external pressure of the barostat can be specified as either a scalar pressure (isobaric ensemble) or as components of a symmetric stress tensor (constant stress ensemble). When used correctly, the time−averaged temperature and stress tensor of the particles will match the target values specified by Tstart/Tstop and Pstart/Pstop.

The operation of these fixes is exactly like that described by the fix nvt, npt, and nph commands, except that the radius and radial velocity of electrons are also updated. Likewise the temperature and pressure calculated by the fix, using the computes it creates (as discussed in the fix nvt, npt, and nph doc page), are performed with computes that include the eFF contribution to the temperature or kinetic energy from the electron radial velocity.

IMPORTANT NOTE: there are two different pressures that can be reported for eFF when defining the pair_style (see pair eff/cut to understand these settings), one (default) that considers electrons do not contribute radial virial components (i.e. electrons treated as incompressible 'rigid' spheres) and one that does. The radial electronic contributions to the virials are only tallied if the flexible pressure option is set, and this will affect both global and per−atom quantities. In principle, the true pressure of a system is somewhere in between the rigid and the flexible eFF pressures, but, for most cases, the difference between these two pressures will not be significant over long−term averaged runs (i.e. even though the energy partitioning changes, the total energy remains similar).

IMPORTANT NOTE: currently, there is no available option for the user to set or create temperature distributions that include the radial electronic degrees of freedom with the velocity command, so the the user must allow for these degrees of freedom to equilibrate (i.e. equi−partitioning of energy) through time integration.

Restart, fix_modify, output, run start/stop, minimize info:

See the doc page for the fix nvt, npt, and nph commands for details.

Restrictions:

This fix is part of the "user−eff" package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

Other restriction discussed on the doc page for the fix nvt, npt, and nph commands also apply.

IMPORTANT NOTE: The temperature for systems (regions or groups) with only electrons and no nuclei is 0.0 (i.e. not defined) in the current temperature calculations, a practical example would be a uniform electron gas or a very hot plasma, where electrons remain delocalized from the nuclei. This is because, even though electron virials are included in the temperature calculation, these are averaged over the nuclear degrees of freedom only. In such cases a corrective term must be added to the pressure to get the correct kinetic contribution.

Related commands:

fix nvt, fix nph, fix npt, fix_modify, run_style

Default:

The keyword defaults are tchain = 3, pchain = 3, mtk = yes, tloop = ploop = 1, nreset = 0, drag = 0.0, dilate = all, and couple = none.

(Martyna) Martyna, Tobias and Klein, J Chem Phys, 101, 4177 (1994).

(Parrinello) Parrinello and Rahman, J Appl Phys, 52, 7182 (1981).

(Tuckerman) Tuckerman, Alejandre, Lopez−Rendon, Jochim, and Martyna, J Phys A: Math Gen, 39, 5629 (2006).

(Shinoda) Shinoda, Shiga, and Mikami, Phys Rev B, 69, 134103 (2004).

fix nph/asphere command

Syntax:

```
fix ID group-ID nph/asphere args keyword value ...
```

- ID, group−ID are documented in fix command
- nph/asphere = style name of this fix command
- additional barostat related keyword/value pairs from the fix nph command can be appended

Examples:

```
fix 1 all nph/asphere iso 0.0 0.0 1000.0
fix 2 all nph/asphere x 5.0 5.0 1000.0
fix 2 all nph/asphere x 5.0 5.0 1000.0 drag 0.2
fix 2 water nph/asphere aniso 0.0 0.0 1000.0 dilate partial
```

Description:

Perform constant NPH integration to update position, velocity, orientation, and angular velocity each timestep for aspherical or ellipsoidal particles in the group using a Nose/Hoover pressure barostat. P is pressure; H is enthalpy. This creates a system trajectory consistent with the isenthalpic ensemble.

This fix differs from the fix nph command, which assumes point particles and only updates their position and velocity.

Additional parameters affecting the barostat are specified by keywords and values documented with the fix nph command. See, for example, discussion of the *aniso*, and *dilate* keywords.

The particles in the fix group are the only ones whose velocities and positions are updated by the velocity/position update portion of the NPH integration.

Regardless of what particles are in the fix group, a global pressure is computed for all particles. Similarly, when the size of the simulation box is changed, all particles are re−scaled to new positions, unless the keyword *dilate* is specified with a value of *partial*, in which case only the particles in the fix group are re−scaled. The latter can be useful for leaving the coordinates of particles in a solid substrate unchanged and controlling the pressure of a surrounding fluid.

This fix computes a temperature and pressure each timestep. To do this, the fix creates its own computes of style "temp/asphere" and "pressure", as if these commands had been issued:

```
compute fix-ID_temp all temp/asphere
compute fix-ID_press all pressure fix-ID_temp
```

See the compute temp/asphere and compute pressure commands for details. Note that the IDs of the new computes are the fix−ID + underscore + "temp" or fix_ID + underscore + "press", and the group for the new computes is "all" since pressure is computed for the entire system.

Note that these are NOT the computes used by thermodynamic output (see the thermo_style command) with ID = *thermo_temp* and *thermo_press*. This means you can change the attributes of this fix's temperature or pressure via the compute_modify command or print this temperature or pressure during thermodynamic output via the

thermo_style custom command using the appropriate compute−ID. It also means that changing attributes of *thermo_temp* or *thermo_press* will have no effect on this fix.

Restart, fix_modify, output, run start/stop, minimize info:

This fix writes the state of the Nose/Hoover barostat to binary restart files. See the read_restart command for info on how to re−specify a fix in an input script that reads a restart file, so that the operation of the fix continues in an uninterrupted fashion.

The fix_modify *temp* and *press* options are supported by this fix. You can use them to assign a compute you have defined to this fix which will be used in its thermostatting or barostatting procedure. If you do this, note that the kinetic energy derived from the compute temperature should be consistent with the virial term computed using all atoms for the pressure. LAMMPS will warn you if you choose to compute temperature on a subset of atoms.

The fix_modify *energy* option is supported by this fix to add the energy change induced by Nose/Hoover barostatting to the system's potential energy as part of thermodynamic output.

This fix computes the same global scalar and global vector of quantities as does the fix nph command.

This fix can ramp its target pressure over multiple runs, using the *start* and *stop* keywords of the run command. See the run command for details of how to do this.

This fix is not invoked during energy minimization.

Restrictions:

This fix is part of the "asphere" package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

This fix requires that atoms store torque and angular momementum and a quaternion as defined by the atom_style ellipsoid command.

All particles in the group must be finite−size. They cannot be point particles, but they can be aspherical or spherical as defined by their shape attribute.

Related commands:

fix nph, fix nve_asphere, fix nvt_asphere, fix npt_asphere, fix_modify

Default: none

fix nph/sphere command

Syntax:

```
fix ID group-ID nph/sphere args keyword value ...
```

- ID, group−ID are documented in fix command
- nph/sphere = style name of this fix command
- additional barostat related keyword/value pairs from the fix nph command can be appended

Examples:

```
fix 1 all nph/sphere iso 0.0 0.0 1000.0
fix 2 all nph/sphere x 5.0 5.0 1000.0
fix 2 all nph/sphere x 5.0 5.0 1000.0 drag 0.2
fix 2 water nph/sphere aniso 0.0 0.0 1000.0 dilate partial
```

Description:

Perform constant NPH integration to update position, velocity, and angular velocity each timestep for extended spherical particles in the group using a Nose/Hoover pressure barostat. P is pressure; H is enthalpy. This creates a system trajectory consistent with the isenthalpic ensemble.

This fix differs from the fix nph command, which assumes point particles and only updates their position and velocity.

Additional parameters affecting the barostat are specified by keywords and values documented with the fix nph command. See, for example, discussion of the *aniso*, and *dilate* keywords.

The particles in the fix group are the only ones whose velocities and positions are updated by the velocity/position update portion of the NPH integration.

Regardless of what particles are in the fix group, a global pressure is computed for all particles. Similarly, when the size of the simulation box is changed, all particles are re−scaled to new positions, unless the keyword *dilate* is specified with a value of *partial*, in which case only the particles in the fix group are re−scaled. The latter can be useful for leaving the coordinates of particles in a solid substrate unchanged and controlling the pressure of a surrounding fluid.

This fix computes a temperature and pressure each timestep. To do this, the fix creates its own computes of style "temp/sphere" and "pressure", as if these commands had been issued:

```
compute fix-ID_temp all temp/sphere
compute fix-ID_press all pressure fix-ID_temp
```

See the compute temp/sphere and compute pressure commands for details. Note that the IDs of the new computes are the fix−ID + underscore + "temp" or fix_ID + underscore + "press", and the group for the new computes is "all" since pressure is computed for the entire system.

Note that these are NOT the computes used by thermodynamic output (see the thermo_style command) with ID = *thermo_temp* and *thermo_press*. This means you can change the attributes of this fix's temperature or pressure via the compute_modify command or print this temperature or pressure during thermodynamic output via the

thermo_style custom command using the appropriate compute−ID. It also means that changing attributes of *thermo_temp* or *thermo_press* will have no effect on this fix.

Restart, fix_modify, output, run start/stop, minimize info:

This fix writes the state of the Nose/Hoover barostat to binary restart files. See the read_restart command for info on how to re−specify a fix in an input script that reads a restart file, so that the operation of the fix continues in an uninterrupted fashion.

The fix_modify *temp* and *press* options are supported by this fix. You can use them to assign a compute you have defined to this fix which will be used in its thermostatting or barostatting procedure. If you do this, note that the kinetic energy derived from the compute temperature should be consistent with the virial term computed using all atoms for the pressure. LAMMPS will warn you if you choose to compute temperature on a subset of atoms.

The fix_modify *energy* option is supported by this fix to add the energy change induced by Nose/Hoover barostatting to the system's potential energy as part of thermodynamic output.

This fix computes the same global scalar and global vector of quantities as does the fix nph command.

This fix can ramp its target pressure over multiple runs, using the *start* and *stop* keywords of the run command. See the run command for details of how to do this.

This fix is not invoked during energy minimization.

Restrictions:

This fix requires that atoms store torque and angular velocity (omega) and a radius as defined by the atom_style sphere command.

All particles in the group must be finite−size spheres. They cannot be point particles.

Related commands:

fix nph, fix nve_sphere, fix nvt_sphere, fix npt_sphere, fix_modify

Default: none

fix npt/asphere command

Syntax:

```
fix ID group-ID npt/asphere keyword value ...
```

- ID, group−ID are documented in fix command
- npt/asphere = style name of this fix command
- additional thermostat and barostat related keyword/value pairs from the fix npt command can be appended

Examples:

```
fix 1 all npt/asphere temp 300.0 300.0 100.0 iso 0.0 0.0 1000.0
fix 2 all npt/asphere temp 300.0 300.0 100.0 x 5.0 5.0 1000.0
fix 2 all npt/asphere temp 300.0 300.0 100.0 x 5.0 5.0 1000.0 drag 0.2
fix 2 water npt/asphere temp 300.0 300.0 100.0 aniso 0.0 0.0 1000.0 dilate partial
```

Description:

Perform constant NPT integration to update position, velocity, orientation, and angular velocity each timestep for aspherical or ellipsoidal particles in the group using a Nose/Hoover temperature thermostat and Nose/Hoover pressure barostat. P is pressure; T is temperature. This creates a system trajectory consistent with the isothermal−isobaric ensemble.

This fix differs from the fix npt command, which assumes point particles and only updates their position and velocity.

The thermostat is applied to both the translational and rotational degrees of freedom for the aspherical particles, assuming a compute is used which calculates a temperature that includes the rotational degrees of freedom (see below). The translational degrees of freedom can also have a bias velocity removed from them before thermostatting takes place; see the description below.

Additional parameters affecting the thermostat and barostat are specified by keywords and values documented with the fix npt command. See, for example, discussion of the *temp*, *iso*, *aniso*, and *dilate* keywords.

The particles in the fix group are the only ones whose velocities and positions are updated by the velocity/position update portion of the NPT integration.

Regardless of what particles are in the fix group, a global pressure is computed for all particles. Similarly, when the size of the simulation box is changed, all particles are re−scaled to new positions, unless the keyword *dilate* is specified with a value of *partial*, in which case only the particles in the fix group are re−scaled. The latter can be useful for leaving the coordinates of particles in a solid substrate unchanged and controlling the pressure of a surrounding fluid.

This fix computes a temperature and pressure each timestep. To do this, the fix creates its own computes of style "temp/asphere" and "pressure", as if these commands had been issued:

```
compute fix-ID_temp all temp/asphere
compute fix-ID_press all pressure fix-ID_temp
```

See the compute temp/asphere and compute pressure commands for details. Note that the IDs of the new computes are the fix−ID + underscore + "temp" or fix_ID + underscore + "press", and the group for the new computes is "all" since pressure is computed for the entire system.

Note that these are NOT the computes used by thermodynamic output (see the thermo_style command) with ID = *thermo_temp* and *thermo_press*. This means you can change the attributes of this fix's temperature or pressure via the compute_modify command or print this temperature or pressure during thermodynamic output via the thermo_style custom command using the appropriate compute−ID. It also means that changing attributes of *thermo_temp* or *thermo_press* will have no effect on this fix.

Like other fixes that perform thermostatting, this fix can be used with compute commands that calculate a temperature after removing a "bias" from the atom velocities. E.g. removing the center−of−mass velocity from a group of atoms or only calculating temperature on the x−component of velocity or only calculating temperature for atoms in a geometric region. This is not done by default, but only if the fix_modify command is used to assign a temperature compute to this fix that includes such a bias term. See the doc pages for individual compute commands to determine which ones include a bias. In this case, the thermostat works in the following manner: the current temperature is calculated taking the bias into account, bias is removed from each atom, thermostatting is performed on the remaining thermal degrees of freedom, and the bias is added back in.

Restart, fix_modify, output, run start/stop, minimize info:

This fix writes the state of the Nose/Hoover thermostat and barostat to binary restart files. See the read_restart command for info on how to re−specify a fix in an input script that reads a restart file, so that the operation of the fix continues in an uninterrupted fashion.

The fix_modify *temp* and *press* options are supported by this fix. You can use them to assign a compute you have defined to this fix which will be used in its thermostatting or barostatting procedure. If you do this, note that the kinetic energy derived from the compute temperature should be consistent with the virial term computed using all atoms for the pressure. LAMMPS will warn you if you choose to compute temperature on a subset of atoms.

The fix_modify *energy* option is supported by this fix to add the energy change induced by Nose/Hoover thermostatting and barostatting to the system's potential energy as part of thermodynamic output.

This fix computes the same global scalar and global vector of quantities as does the fix npt command.

This fix can ramp its target temperature and pressure over multiple runs, using the *start* and *stop* keywords of the run command. See the run command for details of how to do this.

This fix is not invoked during energy minimization.

Restrictions:

This fix is part of the "asphere" package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

This fix requires that atoms store torque and angular momementum and a quaternion as defined by the atom_style ellipsoid command.

All particles in the group must be finite−size. They cannot be point particles, but they can be aspherical or spherical as defined by their shape attribute.

Related commands:

fix npt, fix nve_asphere, fix nvt_asphere, fix_modify

Default: none

fix npt/sphere command

Syntax:

```
fix ID group-ID npt/sphere keyword value ...
```

- ID, group−ID are documented in fix command
- npt/sphere = style name of this fix command
- additional thermostat and barostat related keyword/value pairs from the fix npt command can be appended

Examples:

```
fix 1 all npt/sphere temp 300.0 300.0 100.0 iso 0.0 0.0 1000.0
fix 2 all npt/sphere temp 300.0 300.0 100.0 x 5.0 5.0 1000.0
fix 2 all npt/sphere temp 300.0 300.0 100.0 x 5.0 5.0 1000.0 drag 0.2
fix 2 water npt/sphere temp 300.0 300.0 100.0 aniso 0.0 0.0 1000.0 dilate partial
```

Description:

Perform constant NPT integration to update position, velocity, and angular velocity each timestep for extended spherical particles in the group using a Nose/Hoover temperature thermostat and Nose/Hoover pressure barostat. P is pressure; T is temperature. This creates a system trajectory consistent with the isothermal−isobaric ensemble.

This fix differs from the fix npt command, which assumes point particles and only updates their position and velocity.

The thermostat is applied to both the translational and rotational degrees of freedom for the spherical particles, assuming a compute is used which calculates a temperature that includes the rotational degrees of freedom (see below). The translational degrees of freedom can also have a bias velocity removed from them before thermostatting takes place; see the description below.

Additional parameters affecting the thermostat and barostat are specified by keywords and values documented with the fix npt command. See, for example, discussion of the *temp*, *iso*, *aniso*, and *dilate* keywords.

The particles in the fix group are the only ones whose velocities and positions are updated by the velocity/position update portion of the NPT integration.

Regardless of what particles are in the fix group, a global pressure is computed for all particles. Similarly, when the size of the simulation box is changed, all particles are re−scaled to new positions, unless the keyword *dilate* is specified with a value of *partial*, in which case only the particles in the fix group are re−scaled. The latter can be useful for leaving the coordinates of particles in a solid substrate unchanged and controlling the pressure of a surrounding fluid.

This fix computes a temperature and pressure each timestep. To do this, the fix creates its own computes of style "temp/sphere" and "pressure", as if these commands had been issued:

```
compute fix-ID_temp all temp/sphere
compute fix-ID_press all pressure fix-ID_temp
```

See the compute temp/sphere and compute pressure commands for details. Note that the IDs of the new computes are the fix−ID + underscore + "temp" or fix_ID + underscore + "press", and the group for the new computes is "all" since pressure is computed for the entire system.

Note that these are NOT the computes used by thermodynamic output (see the thermo_style command) with ID = *thermo_temp* and *thermo_press*. This means you can change the attributes of this fix's temperature or pressure via the compute_modify command or print this temperature or pressure during thermodynamic output via the thermo_style custom command using the appropriate compute−ID. It also means that changing attributes of *thermo_temp* or *thermo_press* will have no effect on this fix.

Like other fixes that perform thermostatting, this fix can be used with compute commands that calculate a temperature after removing a "bias" from the atom velocities. E.g. removing the center−of−mass velocity from a group of atoms or only calculating temperature on the x−component of velocity or only calculating temperature for atoms in a geometric region. This is not done by default, but only if the fix_modify command is used to assign a temperature compute to this fix that includes such a bias term. See the doc pages for individual compute commands to determine which ones include a bias. In this case, the thermostat works in the following manner: the current temperature is calculated taking the bias into account, bias is removed from each atom, thermostatting is performed on the remaining thermal degrees of freedom, and the bias is added back in.

Restart, fix_modify, output, run start/stop, minimize info:

This fix writes the state of the Nose/Hoover thermostat and barostat to binary restart files. See the read_restart command for info on how to re−specify a fix in an input script that reads a restart file, so that the operation of the fix continues in an uninterrupted fashion.

The fix_modify *temp* and *press* options are supported by this fix. You can use them to assign a compute you have defined to this fix which will be used in its thermostatting or barostatting procedure. If you do this, note that the kinetic energy derived from the compute temperature should be consistent with the virial term computed using all atoms for the pressure. LAMMPS will warn you if you choose to compute temperature on a subset of atoms.

The fix_modify *energy* option is supported by this fix to add the energy change induced by Nose/Hoover thermostatting and barostatting to the system's potential energy as part of thermodynamic output.

This fix computes the same global scalar and global vector of quantities as does the fix npt command.

This fix can ramp its target temperature and pressure over multiple runs, using the *start* and *stop* keywords of the run command. See the run command for details of how to do this.

This fix is not invoked during energy minimization.

Restrictions:

This fix requires that atoms store torque and angular velocity (omega) and a radius as defined by the atom_style sphere command.

All particles in the group must be finite−size spheres. They cannot be point particles.

Related commands:

fix npt, fix nve_sphere, fix nvt_sphere, fix npt_asphere, fix_modify

Default: none

fix nve command

fix nve/cuda command

Syntax:

```
fix ID group-ID nve
```

- ID, group−ID are documented in fix command
- nve = style name of this fix command

Examples:

```
fix 1 all nve
```

Description:

Perform constant NVE integration to update position and velocity for atoms in the group each timestep. V is volume; E is energy. This creates a system trajectory consistent with the microcanonical ensemble.

Styles with a *cuda* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in this section of the manual. The accelerated styles take the same arguments and should produce the same results, except for round−off and precision issues.

These accelerated styles are part of the "user−cuda" package. They are only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the −suffix command−line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See this section of the manual for more instructions on how to use the accelerated styles effectively.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix. No global or per−atom quantities are stored by this fix for access by various output commands. No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions: none

Related commands:

fix nvt, fix npt

Default: none

fix nve/asphere command

Syntax:

```
fix ID group-ID nve/asphere
```

- ID, group−ID are documented in fix command
- nve/asphere = style name of this fix command

Examples:

```
fix 1 all nve/asphere
```

Description:

Perform constant NVE integration to update position, velocity, orientation, and angular velocity for aspherical particles in the group each timestep. V is volume; E is energy. This creates a system trajectory consistent with the microcanonical ensemble.

This fix differs from the fix nve command, which assumes point particles and only updates their position and velocity.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix. No global or per−atom quantities are stored by this fix for access by various output commands. No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions:

This fix is part of the "asphere" package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

This fix requires that atoms store torque and angular momementum and a quaternion as defined by the atom_style ellipsoid command.

All particles in the group must be finite−size. They cannot be point particles, but they can be aspherical or spherical as defined by their shape attribute.

Related commands:

fix nve, fix nve/sphere

Default: none

fix nve/eff command

Syntax:

```
fix ID group-ID nve/eff
```

- ID, group−ID are documented in fix command
- nve/eff = style name of this fix command

Examples:

```
fix 1 all nve/eff
```

Description:

Perform constant NVE integration to update position and velocity for nuclei and electrons in the group for the electron force field model. V is volume; E is energy. This creates a system trajectory consistent with the microcanonical ensemble.

The operation of this fix is exactly like that described by the fix nve command, except that the radius and radial velocity of electrons are also updated.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix. No global or per−atom quantities are stored by this fix for access by various output commands. No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions:

This fix is part of the "user−eff" package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

Related commands:

fix nve, fix nvt/eff, fix npt/eff

Default: none

fix nve/limit command

Syntax:

```
fix ID group-ID nve/limit xmax
```

- ID, group−ID are documented in fix command
- nve = style name of this fix command
- xmax = maximum distance an atom can move in one timestep (distance units)

Examples:

```
fix 1 all nve/limit 0.1
```

Description:

Perform constant NVE updates of position and velocity for atoms in the group each timestep. A limit is imposed on the maximum distance an atom can move in one timestep. This is useful when starting a simulation with a configuration containing highly overlapped atoms. Normally this would generate huge forces which would blow atoms out of the simulation box, causing LAMMPS to stop with an error.

Using this fix can overcome that problem. Forces on atoms must still be computable (which typically means 2 atoms must have a separation distance > 0.0). But large velocities generated by large forces are reset to a value that corresponds to a displacement of length *xmax* in a single timestep. *Xmax* is specified in distance units; see the units command for details. The value of *xmax* should be consistent with the neighbor skin distance and the frequency of neighbor list re−building, so that pairwise interactions are not missed on successive timesteps as atoms move. See the neighbor and neigh_modify commands for details.

Note that if a velocity reset occurs the integrator will not conserve energy. On steps where no velocity resets occur, this integrator is exactly like the fix nve command. Since forces are unaltered, pressures computed by thermodynamic output will still be very large for overlapped configurations.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix.

This fix computes a global scalar which can be accessed by various output commands. The scalar is the count of how many updates of atom's velocity/position were limited by the maximum distance criterion. This should be roughly the number of atoms so affected, except that updates occur at both the beginning and end of a timestep in a velocity Verlet timestepping algorithm. This is a cumulative quantity for the current run, but is re−initialized to zero each time a run is performed. The scalar value calculated by this fix is "extensive".

No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions: none

Related commands:

fix nve, fix nve/noforce, pair_style soft

Default: none

fix nve/noforce command

Syntax:

```
fix ID group-ID nve
```

- ID, group−ID are documented in fix command
- nve/noforce = style name of this fix command

Examples:

```
fix 3 wall nve/noforce
```

Description:

Perform updates of position, but not velocity for atoms in the group each timestep. In other words, the force on the atoms is ignored and their velocity is not updated. The atom velocities are used to update their positions.

This can be useful for wall atoms, when you set their velocities, and want the wall to move (or stay stationary) in a prescribed fashion.

This can also be accomplished via the fix setforce command, but with fix nve/noforce, the forces on the wall atoms are unchanged, and can thus be printed by the dump command or queried with an equal−style variable that uses the fcm() group function to compute the total force on the group of atoms.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix. No global or per−atom quantities are stored by this fix for access by various output commands. No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions: none

Related commands:

fix nve

Default: none

fix nve/sphere command

Syntax:

```
fix ID group-ID nve/sphere
```

- ID, group−ID are documented in fix command
- nve/sphere = style name of this fix command
- zero or more keyword/value pairs may be appended
- keyword = *update*

```
update value = dipole
    dipole = update orientation of dipole moment during integration
```

Examples:

```
fix 1 all nve/sphere
fix 1 all nve/sphere update dipole
```

Description:

Perform constant NVE integration to update position, velocity, and angular velocity for extended spherical particles in the group each timestep. V is volume; E is energy. This creates a system trajectory consistent with the microcanonical ensemble.

This fix differs from the fix nve command, which assumes point particles and only updates their position and velocity.

If the *update* keyword is used with the *dipole* value, then the orientation of the dipole moment of each particle is also updated during the time integration. This option should be used for models where a dipole moment is assigned to particles via use of the atom_style dipole command.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix. No global or per−atom quantities are stored by this fix for access by various output commands. No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions:

This fix requires that atoms store torque and angular velocity (omega) and a radius as defined by the atom_style sphere command. If the *dipole* keyword is used, then they must also store a dipole moment as defined by the atom_style dipole command.

All particles in the group must be finite−size spheres. They cannot be point particles.

Related commands:

fix nve, fix nve/asphere

Default: none

fix nvt/asphere command

Syntax:

```
fix ID group-ID nvt/asphere keyword value ...
```

- ID, group−ID are documented in fix command
- nvt/asphere = style name of this fix command
- additional thermostat related keyword/value pairs from the fix nvt command can be appended

Examples:

```
fix 1 all nvt/asphere temp 300.0 300.0 100.0
fix 1 all nvt/asphere temp 300.0 300.0 100.0 drag 0.2
```

Description:

Perform constant NVT integration to update position, velocity, orientation, and angular velocity each timestep for aspherical or ellipsoidal particles in the group using a Nose/Hoover temperature thermostat. V is volume; T is temperature. This creates a system trajectory consistent with the canonical ensemble.

This fix differs from the fix nvt command, which assumes point particles and only updates their position and velocity.

The thermostat is applied to both the translational and rotational degrees of freedom for the aspherical particles, assuming a compute is used which calculates a temperature that includes the rotational degrees of freedom (see below). The translational degrees of freedom can also have a bias velocity removed from them before thermostatting takes place; see the description below.

Additional parameters affecting the thermostat are specified by keywords and values documented with the fix nvt command. See, for example, discussion of the *temp* and *drag* keywords.

This fix computes a temperature each timestep. To do this, the fix creates its own compute of style "temp/asphere", as if this command had been issued:

```
compute fix-ID_temp group-ID temp/asphere
```

See the compute temp/asphere command for details. Note that the ID of the new compute is the fix−ID + underscore + "temp", and the group for the new compute is the same as the fix group.

Note that this is NOT the compute used by thermodynamic output (see the thermo_style command) with ID = *thermo_temp*. This means you can change the attributes of this fix's temperature (e.g. its degrees−of−freedom) via the compute_modify command or print this temperature during thermodynamic output via the thermo_style custom command using the appropriate compute−ID. It also means that changing attributes of *thermo_temp* will have no effect on this fix.

Like other fixes that perform thermostatting, this fix can be used with compute commands that calculate a temperature after removing a "bias" from the atom velocities. E.g. removing the center−of−mass velocity from a group of atoms or only calculating temperature on the x−component of velocity or only calculating temperature for atoms in a geometric region. This is not done by default, but only if the fix_modify command is used to assign a temperature compute to this fix that includes such a bias term. See the doc pages for individual compute

commands to determine which ones include a bias. In this case, the thermostat works in the following manner: the current temperature is calculated taking the bias into account, bias is removed from each atom, thermostatting is performed on the remaining thermal degrees of freedom, and the bias is added back in.

Restart, fix_modify, output, run start/stop, minimize info:

This fix writes the state of the Nose/Hoover thermostat to binary restart files. See the read_restart command for info on how to re−specify a fix in an input script that reads a restart file, so that the operation of the fix continues in an uninterrupted fashion.

The fix_modify *temp* option is supported by this fix. You can use it to assign a compute you have defined to this fix which will be used in its thermostatting procedure.

The fix_modify *energy* option is supported by this fix to add the energy change induced by Nose/Hoover thermostatting to the system's potential energy as part of thermodynamic output.

This fix computes the same global scalar and global vector of quantities as does the fix nvt command.

This fix can ramp its target temperature over multiple runs, using the *start* and *stop* keywords of the run command. See the run command for details of how to do this.

This fix is not invoked during energy minimization.

Restrictions:

This fix is part of the "asphere" package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

This fix requires that atoms store torque and angular momementum and a quaternion as defined by the atom_style ellipsoid command.

All particles in the group must be finite−size. They cannot be point particles, but they can be aspherical or spherical as defined by their shape attribute.

Related commands:

fix nvt, fix nve_asphere, fix npt_asphere, fix_modify

Default: none

fix nvt/sllod command

Syntax:

```
fix ID group-ID nvt/sllod keyword value ...
```

- ID, group−ID are documented in fix command
- nvt/sllod = style name of this fix command
- additional thermostat related keyword/value pairs from the fix nvt command can be appended

Examples:

```
fix 1 all nvt/sllod temp 300.0 300.0 100.0
fix 1 all nvt/sllod temp 300.0 300.0 100.0 drag 0.2
```

Description:

Perform constant NVT integration to update positions and velocities each timestep for atoms in the group using a Nose/Hoover temperature thermostat. V is volume; T is temperature. This creates a system trajectory consistent with the canonical ensemble.

This thermostat is used for a simulation box that is changing size and/or shape, for example in a non−equilibrium MD (NEMD) simulation. The size/shape change is induced by use of the fix deform command, so each point in the simulation box can be thought of as having a "streaming" velocity. This position−dependent streaming velocity is subtracted from each atom's actual velocity to yield a thermal velocity which is used for temperature computation and thermostatting. For example, if the box is being sheared in x, relative to y, then points at the bottom of the box (low y) have a small x velocity, while points at the top of the box (hi y) have a large x velocity. These velocities do not contribute to the thermal "temperature" of the atom.

IMPORTANT NOTE: Fix deform has an option for remapping either atom coordinates or velocities to the changing simulation box. To use fix nvt/sllod, fix deform should NOT remap atom positions, because fix nvt/sllod adjusts the atom positions and velocities to create a velocity profile that matches the changing box size/shape. Fix deform SHOULD remap atom velocities when atoms cross periodic boundaries since that is consistent with maintaining the velocity profile created by fix nvt/sllod. LAMMPS will give an error if this setting is not consistent.

The SLLOD equations of motion coupled to a Nose/Hoover thermostat are discussed in (Tuckerman) (eqs 4 and 5), which is what is implemented in LAMMPS in a velocity Verlet formulation.

Additional parameters affecting the thermostat are specified by keywords and values documented with the fix nvt command. See, for example, discussion of the *temp* and *drag* keywords.

This fix computes a temperature each timestep. To do this, the fix creates its own compute of style "temp/deform", as if this command had been issued:

```
compute fix-ID_temp group-ID temp/deform
```

See the compute temp/deform command for details. Note that the ID of the new compute is the fix−ID + underscore + "temp", and the group for the new compute is the same as the fix group.

Note that this is NOT the compute used by thermodynamic output (see the thermo_style command) with ID = *thermo_temp*. This means you can change the attributes of this fix's temperature (e.g. its degrees−of−freedom) via the compute_modify command or print this temperature during thermodynamic output via the thermo_style custom command using the appropriate compute−ID. It also means that changing attributes of *thermo_temp* will have no effect on this fix.

Like other fixes that perform thermostatting, this fix can be used with compute commands that calculate a temperature after removing a "bias" from the atom velocities. E.g. removing the center−of−mass velocity from a group of atoms or only calculating temperature on the x−component of velocity or only calculating temperature for atoms in a geometric region. This is not done by default, but only if the fix_modify command is used to assign a temperature compute to this fix that includes such a bias term. See the doc pages for individual compute commands to determine which ones include a bias. In this case, the thermostat works in the following manner: the current temperature is calculated taking the bias into account, bias is removed from each atom, thermostatting is performed on the remaining thermal degrees of freedom, and the bias is added back in.

Restart, fix_modify, output, run start/stop, minimize info:

This fix writes the state of the Nose/Hoover thermostat to binary restart files. See the read_restart command for info on how to re−specify a fix in an input script that reads a restart file, so that the operation of the fix continues in an uninterrupted fashion.

The fix_modify *temp* option is supported by this fix. You can use it to assign a compute you have defined to this fix which will be used in its thermostatting procedure.

The fix_modify *energy* option is supported by this fix to add the energy change induced by Nose/Hoover thermostatting to the system's potential energy as part of thermodynamic output.

This fix computes the same global scalar and global vector of quantities as does the fix nvt command.

This fix can ramp its target temperature over multiple runs, using the *start* and *stop* keywords of the run command. See the run command for details of how to do this.

This fix is not invoked during energy minimization.

Restrictions:

This fix works best without Nose−Hoover chain thermostats, i.e. using tchain = 1. Setting tchain to larger values can result in poor equilibration.

Related commands:

fix nve, fix nvt, fix temp/rescale, fix langevin, fix_modify, compute temp/deform

Default:

Same as fix nvt, except tchain = 1.

(Tuckerman) Tuckerman, Mundy, Balasubramanian, Klein, J Chem Phys, 106, 5615 (1997).

fix nvt/sllod/eff command

Syntax:

```
fix ID group−ID nvt/sllod/eff keyword value ...
```

- ID, group−ID are documented in fix command
- nvt/sllod/eff = style name of this fix command
- additional thermostat related keyword/value pairs from the fix nvt/eff command can be appended

Examples:

```
fix 1 all nvt/sllod/eff temp 300.0 300.0 0.1
fix 1 all nvt/sllod/eff temp 300.0 300.0 0.1 drag 0.2
```

Description:

Perform constant NVT integration to update positions and velocities each timestep for nuclei and electrons in the group for the electron force field model, using a Nose/Hoover temperature thermostat. V is volume; T is temperature. This creates a system trajectory consistent with the canonical ensemble.

The operation of this fix is exactly like that described by the fix nvt/sllod command, except that the radius and radial velocity of electrons are also updated and thermostatted. Likewise the temperature calculated by the fix, using the compute it creates (as discussed in the fix nvt, npt, and nph doc page), is performed with a compute temp/deform/eff commmand that includes the eFF contribution to the temperature from the electron radial velocity.

Restart, fix_modify, output, run start/stop, minimize info:

This fix writes the state of the Nose/Hoover thermostat to binary restart files. See the read_restart command for info on how to re−specify a fix in an input script that reads a restart file, so that the operation of the fix continues in an uninterrupted fashion.

The fix_modify *temp* option is supported by this fix. You can use it to assign a compute you have defined to this fix which will be used in its thermostatting procedure.

The fix_modify *energy* option is supported by this fix to add the energy change induced by Nose/Hoover thermostatting to the system's potential energy as part of thermodynamic output.

This fix computes the same global scalar and global vector of quantities as does the fix nvt/eff command.

This fix can ramp its target temperature over multiple runs, using the *start* and *stop* keywords of the run command. See the run command for details of how to do this.

This fix is not invoked during energy minimization.

Restrictions:

This fix is part of the "user−eff" package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

This fix works best without Nose−Hoover chain thermostats, i.e. using tchain = 1. Setting tchain to larger values can result in poor equilibration.

Related commands:

fix nve/eff, fix nvt/eff, fix langevin/eff, fix nvt/sllod, fix_modify, compute temp/deform/eff

Default:

Same as fix nvt/eff, except tchain = 1.

(Tuckerman) Tuckerman, Mundy, Balasubramanian, Klein, J Chem Phys, 106, 5615 (1997).

fix nvt/sphere command

Syntax:

```
fix ID group-ID nvt/sphere keyword value ...
```

- ID, group−ID are documented in fix command
- nvt/sphere = style name of this fix command
- additional thermostat related keyword/value pairs from the fix nvt command can be appended

Examples:

```
fix 1 all nvt/sphere temp 300.0 300.0 100.0
fix 1 all nvt/sphere temp 300.0 300.0 100.0 drag 0.2
```

Description:

Perform constant NVT integration to update position, velocity, and angular velocity each timestep for extended spherical particles in the group using a Nose/Hoover temperature thermostat. V is volume; T is temperature. This creates a system trajectory consistent with the canonical ensemble.

This fix differs from the fix nvt command, which assumes point particles and only updates their position and velocity.

The thermostat is applied to both the translational and rotational degrees of freedom for the spherical particles, assuming a compute is used which calculates a temperature that includes the rotational degrees of freedom (see below). The translational degrees of freedom can also have a bias velocity removed from them before thermostatting takes place; see the description below.

Additional parameters affecting the thermostat are specified by keywords and values documented with the fix nvt command. See, for example, discussion of the *temp* and *drag* keywords.

This fix computes a temperature each timestep. To do this, the fix creates its own compute of style "temp/sphere", as if this command had been issued:

```
compute fix-ID_temp group-ID temp/sphere
```

See the compute temp/sphere command for details. Note that the ID of the new compute is the fix−ID + underscore + "temp", and the group for the new compute is the same as the fix group.

Note that this is NOT the compute used by thermodynamic output (see the thermo_style command) with ID = *thermo_temp*. This means you can change the attributes of this fix's temperature (e.g. its degrees−of−freedom) via the compute_modify command or print this temperature during thermodynamic output via the thermo_style custom command using the appropriate compute−ID. It also means that changing attributes of *thermo_temp* will have no effect on this fix.

Like other fixes that perform thermostatting, this fix can be used with compute commands that calculate a temperature after removing a "bias" from the atom velocities. E.g. removing the center−of−mass velocity from a group of atoms or only calculating temperature on the x−component of velocity or only calculating temperature for atoms in a geometric region. This is not done by default, but only if the fix_modify command is used to assign a temperature compute to this fix that includes such a bias term. See the doc pages for individual compute

commands to determine which ones include a bias. In this case, the thermostat works in the following manner: the current temperature is calculated taking the bias into account, bias is removed from each atom, thermostatting is performed on the remaining thermal degrees of freedom, and the bias is added back in.

Restart, fix_modify, output, run start/stop, minimize info:

This fix writes the state of the Nose/Hoover thermostat to binary restart files. See the read_restart command for info on how to re−specify a fix in an input script that reads a restart file, so that the operation of the fix continues in an uninterrupted fashion.

The fix_modify *temp* option is supported by this fix. You can use it to assign a compute you have defined to this fix which will be used in its thermostatting procedure.

The fix_modify *energy* option is supported by this fix to add the energy change induced by Nose/Hoover thermostatting to the system's potential energy as part of thermodynamic output.

This fix computes the same global scalar and global vector of quantities as does the fix nvt command.

This fix can ramp its target temperature over multiple runs, using the *start* and *stop* keywords of the run command. See the run command for details of how to do this.

This fix is not invoked during energy minimization.

Restrictions:

This fix requires that atoms store torque and angular velocity (omega) and a radius as defined by the atom_style sphere command.

All particles in the group must be finite−size spheres. They cannot be point particles.

Related commands:

fix nvt, fix nve_sphere, fix nvt_asphere, fix npt_sphere, fix_modify

Default: none

fix orient/fcc command

```
fix ID group−ID orient/fcc nstats dir alat dE cutlo cuthi file0 file1
```

- ID, group−ID are documented in fix command
- nstats = print stats every this many steps, 0 = never
- dir = 0/1 for which crystal is used as reference
- alat = fcc cubic lattice constant (distance units)
- dE = energy added to each atom (energy units)
- cutlo,cuthi = values between 0.0 and 1.0, cutlo < cuthi
- file0,file1 = files that specify orientation of each grain

Examples:

```
fix gb all orient/fcc 0 1 4.032008 0.001 0.25 0.75 xi.vec chi.vec
```

Description:

The fix applies an orientation−dependent force to atoms near a planar grain boundary which can be used to induce grain boundary migration (in the direction perpendicular to the grain boundary plane). The motivation and explanation of this force and its application are described in (Janssens). The force is only applied to atoms in the fix group.

The basic idea is that atoms in one grain (on one side of the boundary) have a potential energy dE added to them. Atoms in the other grain have 0.0 potential energy added. Atoms near the boundary (whose neighbor environment is intermediate between the two grain orientations) have an energy between 0.0 and dE added. This creates an effective driving force to reduce the potential energy of atoms near the boundary by pushing them towards one of the grain orientations. For dir = 1 and dE > 0, the boundary will thus move so that the grain described by file0 grows and the grain described by file1 shrinks. Thus this fix is designed for simulations of two−grain systems, either with one grain boundary and free surfaces parallel to the boundary, or a system with periodic boundary conditions and two equal and opposite grain boundaries. In either case, the entire system can displace during the simulation, and such motion should be accounted for in measuring the grain boundary velocity.

The potential energy added to atom I is given by these formulas

$$\xi_i = \sum_{j=1}^{12} \left| \mathbf{r}_j - \mathbf{r}_j^{\mathrm{I}} \right| \tag{1}$$

$$\xi_{\mathrm{IJ}} = \sum_{j=1}^{12} \left| \mathbf{r}_j^{\mathrm{J}} - \mathbf{r}_j^{\mathrm{I}} \right| \tag{2}$$

$$\xi_{\mathrm{low}} = \mathrm{cutlo}\,\xi_{\mathrm{IJ}} \tag{3}$$

$$\xi_{\mathrm{high}} = \mathrm{cuthi}\,\xi_{\mathrm{IJ}} \tag{4}$$

$$\omega_i = \frac{\pi}{2} \frac{\xi_i - \xi_{\mathrm{low}}}{\xi_{\mathrm{high}} - \xi_{\mathrm{low}}} \tag{5}$$

$$\begin{aligned} u_i &= 0 && \text{for } \xi_i < \xi_{\mathrm{low}} \\ &= \mathrm{dE}\,\frac{1-\cos(2\omega_i)}{2} && \text{for } \xi_{\mathrm{low}} < \xi_i < \xi_{\mathrm{high}} \\ &= \mathrm{dE} && \text{for } \xi_{\mathrm{high}} < \xi_i \end{aligned} \tag{6}$$

which are fully explained in (Janssens). The order parameter Xi for atom I in equation (1) is a sum over the 12 nearest neighbors of atom I. Rj is the vector from atom I to its neighbor J, and RIj is a vector in the reference (perfect) crystal. That is, if dir = 0/1, then RIj is a vector to an atom coord from file 0/1. Equation (2) gives the expected value of the order parameter XiIJ in the other grain. Hi and lo cutoffs are defined in equations (3) and (4), using the input parameters *cutlo* and *cuthi* as thresholds to avoid adding grain boundary energy when the deviation in the order parameter from 0 or 1 is small (e.g. due to thermal fluctuations in a perfect crystal). The added potential energy Ui for atom I is given in equation (6) where it is interpolated between 0 and dE using the two threshold Xi values and the Wi value of equation (5).

The derivative of this energy expression gives the force on each atom which thus depends on the orientation of its neighbors relative to the 2 grain orientations. Only atoms near the grain boundary feel a net force which tends to drive them to one of the two grain orientations.

In equation (1), the reference vector used for each neighbor is the reference vector closest to the actual neighbor position. This means it is possible two different neighbors will use the same reference vector. In such cases, the atom in question is far from a perfect orientation and will likely receive the full dE addition, so the effect of duplicate reference vector usage is small.

The *dir* parameter determines which grain wants to grow at the expense of the other. A value of 0 means the first grain will shrink; a value of 1 means it will grow. This assumes that *dE* is positive. The reverse will be true if *dE* is negative.

The *alat* parameter is the cubic lattice constant for the fcc material and is only used to compute a cutoff distance of 1.57 * alat / sqrt(2) for finding the 12 nearest neighbors of each atom (which should be valid for an fcc crystal). A longer/shorter cutoff can be imposed by adjusting *alat*. If a particular atom has less than 12 neighbors within the cutoff, the order parameter of equation (1) is effectively multiplied by 12 divided by the actual number of neighbors within the cutoff.

The *dE* parameter is the maximum amount of additional energy added to each atom in the grain which wants to shrink.

The *cutlo* and *cuthi* parameters are used to reduce the force added to bulk atoms in each grain far away from the boundary. An atom in the bulk surrounded by neighbors at the ideal grain orientation would compute an order parameter of 0 or 1 and have no force added. However, thermal vibrations in the solid will cause the order parameters to be greater than 0 or less than 1. The cutoff parameters mask this effect, allowing forces to only be added to atoms with order−parameters between the cutoff values.

File0 and *file1* are filenames for the two grains which each contain 6 vectors (6 lines with 3 values per line) which specify the grain orientations. Each vector is a displacement from a central atom (0,0,0) to a nearest neighbor atom in an fcc lattice at the proper orientation. The vector lengths should all be identical since an fcc lattice has a coordination number of 12. Only 6 are listed due to symmetry, so the list must include one from each pair of equal−and−opposite neighbors. A pair of orientation files for a Sigma=5 tilt boundary are show below.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files.

The fix_modify *energy* option is supported by this fix to add the potential energy of atom interactions with the grain boundary driving force to the system's potential energy as part of thermodynamic output.

This fix calculates a global scalar which can be accessed by various output commands. The scalar is the potential energy change due to this fix. The scalar value calculated by this fix is "extensive".

This fix also calculates a per−atom array which can be accessed by various output commands. The array stores the order parameter Xi and normalized order parameter (0 to 1) for each atom. The per−atom values can be accessed on any timestep.

No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions:

This fix should only be used with fcc lattices.

Related commands:

fix_modify

Default: none

(Janssens) Janssens, Olmsted, Holm, Foiles, Plimpton, Derlet, Nature Materials, 5, 124−127 (2006).

For illustration purposes, here are example files that specify a Sigma=5 tilt boundary. This is for a lattice constant of 3.5706 Angs.

file0:

```
 0.798410432046075    1.785300000000000    1.596820864092150
-0.798410432046075    1.785300000000000   -1.596820864092150
 2.395231296138225    0.000000000000000    0.798410432046075
 0.798410432046075    0.000000000000000   -2.395231296138225
 1.596820864092150    1.785300000000000   -0.798410432046075
 1.596820864092150   -1.785300000000000   -0.798410432046075
```

file1:

```
−0.798410432046075    1.785300000000000    1.596820864092150
 0.798410432046075    1.785300000000000   −1.596820864092150
 0.798410432046075    0.000000000000000    2.395231296138225
 2.395231296138225    0.000000000000000   −0.798410432046075
 1.596820864092150    1.785300000000000    0.798410432046075
 1.596820864092150   −1.785300000000000    0.798410432046075
```

fix planeforce command

Syntax:

```
fix ID group-ID planeforce x y z
```

- ID, group−ID are documented in fix command
- lineforce = style name of this fix command
- x y z = 3−vector that is normal to the plane

Examples:

```
fix hold boundary planeforce 1.0 0.0 0.0
```

Description:

Adjust the forces on each atom in the group so that only the components of force in the plane specified by the normal vector (x,y,z) remain. This is done by subtracting out the component of force perpendicular to the plane.

If the initial velocity of the atom is 0.0 (or in the plane), then it should continue to move in the plane thereafter.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix. No global or per−atom quantities are stored by this fix for access by various output commands. No parameter of this fix can be used with the *start/stop* keywords of the run command.

The forces due to this fix are imposed during an energy minimization, invoked by the minimize command.

Restrictions: none

Related commands:

fix lineforce

Default: none

fix poems

Syntax:

```
fix ID group-ID poems keyword values
```

- ID, group−ID are documented in fix command
- poems = style name of this fix command
- keyword = *group* or *file* or *molecule*

```
  group values = list of group IDs
    molecule values = none
    file values = filename
```

Examples:

```
fix 3 fluid poems group clump1 clump2 clump3
fix 3 fluid poems file cluster.list
```

Description:

Treats one or more sets of atoms as coupled rigid bodies. This means that each timestep the total force and torque on each rigid body is computed and the coordinates and velocities of the atoms are updated so that the collection of bodies move as a coupled set. This can be useful for treating a large biomolecule as a collection of connected, coarse−grained particles.

The coupling, associated motion constraints, and time integration is performed by the software package Parallelizable Open source Efficient Multibody Software (POEMS) which computes the constrained rigid−body motion of articulated (jointed) multibody systems (Anderson). POEMS was written and is distributed by Prof Kurt Anderson, his graduate student Rudranarayan Mukherjee, and other members of his group at Rensselaer Polytechnic Institute (RPI). Rudranarayan developed the LAMMPS/POEMS interface. For copyright information on POEMS and other details, please refer to the documents in the poems directory distributed with LAMMPS.

This fix updates the positions and velocities of the rigid atoms with a constant−energy time integration, so you should not update the same atoms via other fixes (e.g. nve, nvt, npt, temp/rescale, langevin).

Each body must have a non−degenerate inertia tensor, which means if must contain at least 3 non−collinear atoms. Which atoms are in which bodies can be defined via several options.

For option *group*, each of the listed groups is treated as a rigid body. Note that only atoms that are also in the fix group are included in each rigid body.

For option *molecule*, each set of atoms in the group with a different molecule ID is treated as a rigid body.

For option *file*, sets of atoms are read from the specified file and each set is treated as a rigid body. Each line of the file specifies a rigid body in the following format:

ID type atom1−ID atom2−ID atom3−ID ...

ID as an integer from 1 to M (the number of rigid bodies). Type is any integer; it is not used by the fix poems command. The remaining arguments are IDs of atoms in the rigid body, each typically from 1 to N (the number of

atoms in the system). Only atoms that are also in the fix group are included in each rigid body. Blank lines and lines that begin with '#' are skipped.

A connection between a pair of rigid bodies is inferred if one atom is common to both bodies. The POEMS solver treats that atom as a spherical joint with 3 degrees of freedom. Currently, a collection of bodies can only be connected by joints as a linear chain. The entire collection of rigid bodies can represent one or more chains. Other connection topologies (tree, ring) are not allowed, but will be added later. Note that if no joints exist, it is more efficient to use the fix rigid command to simulate the system.

When the poems fix is defined, it will print out statistics on the total # of clusters, bodies, joints, atoms involved. A cluster in this context means a set of rigid bodies connected by joints.

For computational efficiency, you should turn off pairwise and bond interactions within each rigid body, as they no longer contribute to the motion. The "neigh_modify exclude" and "delete_bonds" commands can be used to do this if each rigid body is a group.

For computational efficiency, you should only define one fix poems which includes all the desired rigid bodies. LAMMPS will allow multiple poems fixes to be defined, but it is more expensive.

The degrees−of−freedom removed by coupled rigid bodies are accounted for in temperature and pressure computations. Similarly, the rigid body contribution to the pressure virial is also accounted for. The latter is only correct if forces within the bodies have been turned off, and there is only a single fix poems defined.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix. No global or per−atom quantities are stored by this fix for access by various output commands. No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions:

This fix is part of the "poems" package. It is only enabled if LAMMPS was built with that package, which also requires the POEMS library be built and linked with LAMMPS. See the Making LAMMPS section for more info.

Related commands:

fix rigid, delete_bonds, neigh_modify exclude

Default: none

(Anderson) Anderson, Mukherjee, Critchley, Ziegler, and Lipton "POEMS: Parallelizable Open−source Efficient Multibody Software ", Engineering With Computers (2006). (link to paper)

fix pour command

Syntax:

```
fix ID group-ID pour N type seed keyword values ...
```

- ID, group−ID are documented in fix command
- pour = style name of this fix command
- N = # of atoms to insert
- type = atom type to assign to inserted atoms
- seed = random # seed (positive integer)
- one or more keyword/value pairs may be appended to args
- keyword = *region* or *diam* or *dens* or *vol* or *rate* or *vel*

```
  region value = region-ID
      region-ID = ID of region to use as insertion volume
    diam values = lo hi
      lo,hi = range of diameters for inserted particles (distance units)
    dens values = lo hi
      lo,hi = range of densities for inserted particles
    vol values = fraction Nattempt
      fraction = desired volume fraction for filling insertion volume
      Nattempt = max # of insertion attempts per atom
    rate value = V
      V = z velocity (3d) or y velocity (2d) at which
          insertion volume moves (velocity units)
    vel values (3d) = vxlo vxhi vylo vyhi vz
    vel values (2d) = vxlo vxhi vy
      vxlo,vxhi = range of x velocities for inserted particles (velocity units)
      vylo,vyhi = range of y velocities for inserted particles (velocity units)
      vz = z velocity (3d) assigned to inserted particles (velocity units)
      vy = y velocity (2d) assigned to inserted particles (velocity units)
```

Examples:

```
fix 3 all pour 1000 2 29494 region myblock
fix 2 all pour 10000 1 19985583 region disk vol 0.33 100 rate 1.0 diam 0.9 1.1
```

Description:

Insert particles into a granular run every few timesteps within a specified region until N particles have been inserted. This is useful for simulating the pouring of particles into a container under the influence of gravity.

Inserted particles are assigned the specified atom type and are assigned to two groups: the default group "all" and the group specified in the fix pour command (which can also be "all").

This command must use the *region* keyword to define an insertion volume. The specified region must have been previously defined with a region command. It must be of type *block* or a z−axis *cylinder* and must be defined with side = *in*. The cylinder style of region can only be used with 3d simulations.

Each timestep particles are inserted, they are placed randomly inside the insertion volume so as to mimic a stream of poured particles. The larger the volume, the more particles that can be inserted at any one timestep. Particles are inserted again after enough time has elapsed that the previously inserted particles fall out of the insertion volume under the influence of gravity. Insertions continue every so many timesteps until the desired # of particles

has been inserted.

All other keywords are optional with defaults as shown below. The *diam*, *dens*, and *vel* options enable inserted particles to have a range of diameters or densities or xy velocities. The specific values for a particular inserted particle will be chosen randomly and uniformly between the specified bounds. The *vz* or *vy* value for option *vel* assigns a z−velocity (3d) or y−velocity (2d) to each inserted particle.

The *vol* option specifies what volume fraction of the insertion volume will be filled with particles. The higher the value, the more particles are inserted each timestep. Since inserted particles cannot overlap, the maximum volume fraction should be no higher than about 0.6. Each timestep particles are inserted, LAMMPS will make up to a total of M tries to insert the new particles without overlaps, where M = # of inserted particles * Nattempt. If LAMMPS is unsuccessful at completing all insertions, it prints a warning.

The *rate* option moves the insertion volume in the z direction (3d) or y direction (2d). This enables pouring particles from a successively higher height over time.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. This means you must be careful when restarting a pouring simulation, when the restart file was written in the middle of the pouring operation. Specifically, you should use a new fix pour command in the input script for the restarted simulation that continues the operation. You will need to adjust the arguments of the original fix pour command to do this.

Also note that because the state of the random number generator is not saved in restart files, you cannot do "exact" restarts with this fix, where the simulation continues on the same as if no restart had taken place. However, in a statistical sense, a restarted simulation should produce the same behavior if you adjust the fix pour parameters appropriately.

None of the fix_modify options are relevant to this fix. No global or per−atom quantities are stored by this fix for access by various output commands. No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions:

This fix is part of the "granular" package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

For 3d simulations, a gravity fix in the −z direction must be defined for use in conjunction with this fix. For 2d simulations, gravity must be defined in the −y direction.

The specified insertion region cannot be a "dynamic" region, as defined by the region command.

Related commands:

fix_deposit, fix_gravity, region

Default:

The option defaults are diam = 1.0 1.0, dens = 1.0 1.0, vol = 0.25 50, rate = 0.0, vel = 0.0 0.0 0.0 0.0 0.0.

fix press/berendsen command

Syntax:

```
fix ID group−ID press/berendsen keyword value ...
```

- ID, group−ID are documented in fix command
- press/berendsen = style name of this fix command

```
one or more keyword value pairs may be appended
keyword = iso or aniso or x or y or z or couple or dilate or modulus
  iso or aniso values = Pstart Pstop Pdamp
    Pstart,Pstop = scalar external pressure at start/end of run (pressure units)
    Pdamp = pressure damping parameter (time units)
  x or y or z values = Pstart Pstop Pdamp
    Pstart,Pstop = external stress tensor component at start/end of run (pressure units)
    Pdamp = stress damping parameter (time units)
  couple = none or xyz or xy or yz or xz
  modulus value = bulk modulus of system (pressure units)
  dilate value = all or partial
```

Examples:

```
fix 1 all press/berendsen iso 0.0 0.0 1000.0
fix 2 all press/berendsen aniso 0.0 0.0 1000.0 dilate partial
```

Description:

Reset the pressure of the system by using a Berendsen barostat (Berendsen), which rescales the system volume and (optionally) the atoms coordinates within the simulation box every timestep.

Regardless of what atoms are in the fix group, a global pressure is computed for all atoms. Similarly, when the size of the simulation box is changed, all atoms are re−scaled to new positions, unless the keyword *dilate* is specified with a value of *partial*, in which case only the atoms in the fix group are re−scaled. The latter can be useful for leaving the coordinates of atoms in a solid substrate unchanged and controlling the pressure of a surrounding fluid.

IMPORTANT NOTE: Unlike the fix npt or fix nph commands which perform Nose/Hoover barostatting AND time integration, this fix does NOT perform time integration. It only modifies the box size and atom coordinates to effect barostatting. Thus you must use a separate time integration fix, like fix nve or fix nvt to actually update the positions and velocities of atoms. This fix can be used in conjunction with thermostatting fixes to control the temperature, such as fix nvt or fix langevin or fix temp/berendsen.

See this howto section of the manual for a discussion of different ways to compute temperature and perform thermostatting and barostatting.

The barostat is specified using one or more of the *iso*, *aniso*, *x*, *y*, *z*, and *couple* keywords. These keywords give you the ability to specify the 3 diagonal components of an external stress tensor, and to couple various of these components together so that the dimensions they represent are varied together during a constant−pressure simulation. Unlike the fix npt and fix nph commands, this fix cannot be used with triclinic (non−orthogonal) simulation boxes to control all 6 components of the general pressure tensor.

The target pressures for each of the 3 diagonal components of the stress tensor can be specified independently via the *x*, *y*, *z*, keywords, which correspond to the 3 simulation box dimensions. For each component, the external pressure or tensor component at each timestep is a ramped value during the run from *Pstart* to *Pstop*. If a target pressure is specified for a component, then the corresponding box dimension will change during a simulation. For example, if the *y* keyword is used, the y−box length will change. A box dimension will not change if that component is not specified, although you have the option to change that dimension via the fix deform command.

For all barostat keywords, the *Pdamp* parameter determines the time scale on which pressure is relaxed. For example, a value of 1000.0 means to relax the pressure in a timespan of (roughly) 1000 time units (tau or fmsec or psec − see the units command).

IMPORTANT NOTE: The relaxation time is actually also a function of the bulk modulus of the system (inverse of isothermal compressibility). The bulk modulus has units of pressure and is the amount of pressure that would need to be applied (isotropically) to reduce the volume of the system by a factor of 2 (assuming the bulk modulus was a constant, independent of density, which it's not). The bulk modulus can be set via the keyword *modulus*. The *Pdamp* parameter is effectively multiplied by the bulk modulus, so if the pressure is relaxing faster than expected or desired, increasing the bulk modulus has the same effect as increasing *Pdamp*. The converse is also true. LAMMPS does not attempt to guess a correct value of the bulk modulus; it just uses 10.0 as a default value which gives reasonable relaxation for a Lennard−Jones liquid, but will be way off for other materials and way too small for solids. Thus you should experiment to find appropriate values of *Pdamp* and/or the *modulus* when using this fix.

The *couple* keyword allows two or three of the diagonal components of the pressure tensor to be "coupled" together. The value specified with the keyword determines which are coupled. For example, *xz* means the *Pxx* and *Pzz* components of the stress tensor are coupled. *Xyz* means all 3 diagonal components are coupled. Coupling means two things: the instantaneous stress will be computed as an average of the corresponding diagonal components, and the coupled box dimensions will be changed together in lockstep, meaning coupled dimensions will be dilated or contracted by the same percentage every timestep. The *Pstart*, *Pstop*, *Pdamp* parameters for any coupled dimensions must be identical. *Couple xyz* can be used for a 2d simulation; the *z* dimension is simply ignored.

The *iso* and *aniso* keywords are simply shortcuts that are equivalent to specifying several other keywords together.

The keyword *iso* means couple all 3 diagonal components together when pressure is computed (hydrostatic pressure), and dilate/contract the dimensions together. Using "iso Pstart Pstop Pdamp" is the same as specifying these 4 keywords:

```
x Pstart Pstop Pdamp
y Pstart Pstop Pdamp
z Pstart Pstop Pdamp
couple xyz
```

The keyword *aniso* means *x*, *y*, and *z* dimensions are controlled independently using the *Pxx*, *Pyy*, and *Pzz* components of the stress tensor as the driving forces, and the specified scalar external pressure. Using "aniso Pstart Pstop Pdamp" is the same as specifying these 4 keywords:

```
x Pstart Pstop Pdamp
y Pstart Pstop Pdamp
z Pstart Pstop Pdamp
couple none
```

This fix computes a temperature and pressure each timestep. To do this, the fix creates its own computes of style

"temp" and "pressure", as if these commands had been issued:

```
compute fix-ID_temp group-ID temp
compute fix-ID_press group-ID pressure fix-ID_temp
```

See the compute temp and compute pressure commands for details. Note that the IDs of the new computes are the fix−ID + underscore + "temp" or fix_ID + underscore + "press", and the group for the new computes is the same as the fix group.

Note that these are NOT the computes used by thermodynamic output (see the thermo_style command) with ID = *thermo_temp* and *thermo_press*. This means you can change the attributes of this fix's temperature or pressure via the compute_modify command or print this temperature or pressure during thermodynamic output via the thermo_style custom command using the appropriate compute−ID. It also means that changing attributes of *thermo_temp* or *thermo_press* will have no effect on this fix.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files.

The fix_modify *temp* and *press* options are supported by this fix. You can use them to assign a compute you have defined to this fix which will be used in its temperature and pressure calculations. If you do this, note that the kinetic energy derived from the compute temperature should be consistent with the virial term computed using all atoms for the pressure. LAMMPS will warn you if you choose to compute temperature on a subset of atoms.

No global or per−atom quantities are stored by this fix for access by various output commands.

This fix can ramp its target pressure over multiple runs, using the *start* and *stop* keywords of the run command. See the run command for details of how to do this.

This fix is not invoked during energy minimization.

Restrictions:

Any dimension being adjusted by this fix must be periodic.

Related commands:

fix nve, fix nph, fix npt, fix temp/berendsen, fix_modify

Default:

The keyword defaults are dilate = all, modulus = 10.0 in units of pressure for whatever units are defined.

(Berendsen) Berendsen, Postma, van Gunsteren, DiNola, Haak, J Chem Phys, 81, 3684 (1984).

fix print command

Syntax:

```
fix ID group-ID print N string keyword value ...
```

- ID, group−ID are documented in fix command
- print = style name of this fix command
- N = print every N steps
- string = text string to print with optional variable names
- zero or more keyword/value pairs may be appended
- keyword = *file* or *append* or *screen* or *title*

```
file value = filename
  append value = filename
  screen value = yes or no
  title value = string
    string =  text to print as 1st line of output file
```

Examples:

```
fix extra all print 100 "Coords of marker atom = $x $y $z"
fix extra all print 100 "Coords of marker atom = $x $y $z" file coord.txt
```

Description:

Print a text string every N steps during a simulation run. This can be used for diagnostic purposes or as a debugging tool to monitor some quantity during a run. The text string must be a single argument, so it should be enclosed in double quotes if it is more than one word. If it contains variables it must be enclosed in double quotes to insure they are not evaluated when the input script line is read, but will instead be evaluated each time the string is printed.

See the variable command for a description of *equal* style variables which are the most useful ones to use with the fix print command, since they are evaluated afresh each timestep that the fix print line is output. Equal−style variables calculate formulas involving mathematical operations, atom properties, group properties, thermodynamic properties, global values calculated by a compute or fix, or references to other variables.

If the *file* or *append* keyword is used, a filename is specified to which the output generated by this fix will be written. If *file* is used, then the filename is overwritten if it already exists. If *append* is used, then the filename is appended to if it already exists, or created if it does not exist.

If the *screen* keyword is used, output by this fix to the screen and logfile can be turned on or off as desired.

The *title* keyword allow specification of the string that will be printed as the first line of the output file, assuming the *file* keyword was used. By default, the title line is as follows:

```
# Fix print output for fix ID
```

where ID is replaced with the fix−ID.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix. No global or per−atom quantities are stored by this fix for access by various output commands. No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions: none

Related commands:

variable, print

Default:

The option defaults are no file output, screen = yes, and title string as described above.

fix qeq/comb command

Syntax:

```
fix ID group−ID qeq/comb Nevery precision keyword value ...
```

- ID, group−ID are documented in fix command
- qeq/comb = style name of this fix command
- Nevery = perform charge equilibration every this many steps
- precision = convergence criterion for charge equilibration
- zero or more keyword/value pairs may be appended
- keyword = *file*

```
file value = filename
    filename = name of file to write QEQ equilibration info to
```

Examples:

```
fix 1 surface qeq/comb 10 0.0001
```

Description:

Perform charge equilibration (QeQ) in conjunction with the COMB (Charge−Optimized Many−Body) potential as described in (COMB_1) and (COMB_2). It performs the charge equilibration portion of the calculation using the so−called QEq method, whereby the charge on each atom is adjusted to minimize the energy of the system. This fix can only be used with the COMB potential; see the fix qeq/reax command for a QeQ calculation that can be used with any potential.

Only charges on the atoms in the specified group are equilibrated. The fix relies on the pair style (COMB in this case) to calculate the per−atom electronegativity (effective force on the charges). An electronegativity equalization calculation (or QEq) is performed in an interative fashion, which in parallel requires communication at each iteration for processors to exchange charge information about nearby atoms with each other. See Rappe_and_Goddard and Rick_and_Stuart for details.

During a run, charge equilibration is peformed every *Nevery* time steps. Charge equilibration is also always enforced on the first step of each run. The *precision* argument controls the tolerance for the difference in electronegativity for all atoms during charge equilibration. *Precision* is a trade−off between the cost of performing charge equilibration (more iterations) and accuracy.

If the *file* keyword is used, then information about each equilibration calculation is written to the specifed file.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix.

This fix produces a per−atom vector which can be accessed by various output commands. The vector stores the gradient of the charge on each atom. The per−atom values be accessed on any timestep.

No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions:

This fix command currently only supports pair style *comb*.

Related commands:

pair_style comb

Default:

No file output is performed.

(COMB_1) J. Yu, S. B. Sinnott, S. R. Phillpot, Phys Rev B, 75, 085311 (2007),

(COMB_2) T.−R. Shan, B. D. Devine, T. W. Kemper, S. B. Sinnott, S. R. Phillpot, Phys Rev B, 81, 125328 (2010).

(Rappe_and_Goddard) A. K. Rappe, W. A. Goddard, J Phys Chem 95, 3358 (1991).

(Rick_and_Stuart) S. W. Rick, S. J. Stuart, B. J. Berne, J Chem Phys 101, 16141 (1994).

fix qeq/reax command

Syntax:

```
fix ID group−ID qeq/reax Nevery cutlo cuthi tolerance params
```

- ID, group−ID are documented in fix command
- qeq/reax = style name of this fix command
- Nevery = perform QEq every this many steps
- cutlo,cuthi = lo and hi cutoff for Taper radius
- tolerance = precision to which charges will be equilibrated
- params = reax/c or a filename

Examples:

```
fix 1 all qeq/reax 1 0.0 10.0 1.0e−6 reax/c
fix 1 all qeq/reax 1 0.0 10.0 1.0e−6 param.qeq
```

Description:

Perform the charge equilibration (QEq) method as described in (Rappe and Goddard, 1991) and formulated in (Nakano, 1997). It is typically used in conjunction with the ReaxFF force field model as implemented in the pair_style reax/c command, but it can be used with any potential in LAMMPS, so long as it defines and uses charges on each atom. The fix qeq/comb command should be used to perform charge equliibration with the COMB potential. For more technical details about the charge equilibration performed by fix qeq/reax, see the "(Aktulga)" paper.

The QEq method minimizes the electrostatic energy of the system by adjusting the partial charge on individual atoms based on interactions with their neighbors. It reqires some parameters for each atom type. If the *params* setting above is the word "reax/c", then these are extracted from the pair_style reax/c command and the ReaxFF force field file it reads in. If a file name is specified for *params*, then the parameters are taken from the specified file and the file must contain one line for each atom type. The latter form must be used when performing QeQ with a non−ReaxFF potential. Each line should be formatted as follows:

```
itype chi eta gamma
```

where *itype* is the atom type from 1 to Ntypes, *chi* denotes the electronegativity in eV, *eta* denotes the self−Coulomb potential in eV, and *gamma* denotes the valence orbital exponent. Note that these 3 quantities are also in the ReaxFF potential file, except that eta is defined here as twice the eta value in the ReaxFF file. Note that unlike the rest of LAMMPS, the units of this fix are hard−coded to be A, eV, and electronic charge.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. No global scalar or vector or per−atom quantities are stored by this fix for access by various output commands. No parameter of this fix can be used with the *start/stop* keywords of the run command.

This fix is invoked during energy minimization.

Restrictions:

This fix is part of the "user−reaxc" package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

Related commands:

pair_style reax/c

Default: none

(Rappe) Rappe and Goddard III, Journal of Physical Chemistry, 105, 3358−3363 (1991).

(Nakano) Nakano, Computer Physics Communications, 104, 59−69 (1997).

:link(Aktulga) **(Aktulga)** Aktulga, Fogarty, Pandit, Grama, Parallel Computing, to appear (2011).

fix reax/bonds command

Syntax:

```
fix ID group-ID reax/bonds Nevery filename
```

- ID, group−ID are documented in fix command
- reax/bonds = style name of this fix command
- Nevery = output interval in timesteps
- filename = name of output file

Examples:

fix 1 all reax/bonds 100 bonds.tatb

Description:

Write out the bond information computed by the ReaxFF potential specified by pair_style reax. The bond information is written to *filename* on timesteps that are multiples of *Nevery*, including timestep 0.

The format of the output file should be self−explantory.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix. No global or per−atom quantities are stored by this fix for access by various output commands. No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions:

This fix requires that the pair_style reax be invoked. This fix is part of the "reax" package. It is only enabled if LAMMPS was built with that package, which also requires the REAX library be built and linked with LAMMPS. See the Making LAMMPS section for more info.

Related commands:

pair_style reax

Default:

none

fix recenter command

Syntax:

```
fix ID group-ID recenter x y z keyword value ...
```

- ID, group−ID are documented in fix command
- recenter = style name of this fix command
- x,y,z = constrain center−of−mass to these coords (distance units), any coord can also be NULL or INIT (see below)
- zero or more keyword/value pairs may be appended
- keyword = *shift* or *units*

```
  shift value = group-ID
      group-ID = group of atoms whose coords are shifted
    units value = box or lattice or fraction
```

Examples:

```
fix 1 all recenter 0.0 0.5 0.0
fix 1 all recenter INIT INIT NULL
fix 1 all recenter INIT 0.0 0.0 units box
```

Description:

Constrain the center−of−mass position of a group of atoms by adjusting the coordinates of the atoms every timestep. This is simply a small shift that does not alter the dynamics of the system or change the relative coordinates of any pair of atoms in the group. This can be used to insure the entire collection of atoms (or a portion of them) do not drift during the simulation due to random perturbations (e.g. fix langevin thermostatting).

Distance units for the x,y,z values are determined by the setting of the *units* keyword, as discussed below. One or more x,y,z values can also be specified as NULL, which means exclude that dimension from this operation. Or it can be specified as INIT which means to constrain the center−of−mass to its initial value at the beginning of the run.

The center−of−mass (COM) is computed for the group specified by the fix. If the current COM is different than the specified x,y,z, then a group of atoms has their coordinates shifted by the difference. By default the shifted group is also the group specified by the fix. A different group can be shifted by using the *shift* keyword. For example, the COM could be computed on a protein to keep it in the center of the simulation box. But the entire system (protein + water) could be shifted.

If the *units* keyword is set to *box*, then the distance units of x,y,z are defined by the units command − e.g. Angstroms for *real* units. A *lattice* value means the distance units are in lattice spacings. The lattice command must have been previously used to define the lattice spacing. A *fraction* value means a fractional distance between the lo/hi box boundaries, e.g. 0.5 = middle of the box. The default is to use lattice units.

Note that the velocity command can be used to create velocities with zero aggregate linear and/or angular momentum.

IMPORTANT NOTE: This fix performs its operations at the same point in the timestep as other time integration fixes, such as fix nve, fix nvt, or fix npt. Thus fix recenter should normally be the last such fix specified in the

input script, since the adjustments it makes to atom coordinates should come after the changes made by time integration. LAMMPS will warn you if your fixes are not ordered this way.

IMPORTANT NOTE: If you use this fix on a small group of atoms (e.g. a molecule in solvent) without using the *shift* keyword to adjust the positions of all atoms in the system, then the results can be unpredictable. For example, if the molecule is pushed in one direction by the solvent, its velocity will increase. But its coordinates will be recentered, meaning it is pushed back towards the force. Thus over time, the velocity and temperature of the molecule could become very large (though it won't appear to be moving due to the recentering). If you are thermostatting the entire system, then the solvent would be cooled to compensate. A better solution for this simulation scenario is to use the fix spring command to tether the molecule in place.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix. No global or per−atom quantities are stored by this fix for access by various output commands. No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions:

This fix should not be used with an x,y,z setting that causes a large shift in the system on the 1st timestep, due to the requested COM being very different from the initial COM. This could cause atoms to be lost, especially in parallel. Instead, use the displace_atoms command, which can be used to move atoms a large distance.

Related commands:

fix momentum, velocity

Default:

The option defaults are adjust = fix group−ID, and units = lattice.

fix restrain command

Syntax:

```
fix ID group−ID restrain Kstart Kstop keyword value(s)
```

- ID, group−ID are documented in fix command
- restrain = style name of this fix command
- Kstart, Kstop = restraint coefficient at start/end of run (energy units)
- one keyword with one or more sets of parameter values may be appended to args
- keyword = *dihedral*

```
  dihedral value = atom1 atom2 atom3 atom4 target
      atom1,atom2,atom3,atom4 = IDs of 4 atoms in restrained dihedral
      target = target value for specified dihedral angle (degrees)
```

Examples:

```
fix holdem all restrain 2000.0 2000.0 dihedral 1 2 3 4 120.0
fix texas_holdem all restrain 0.0 2000.0 dihedral 1 2 3 4 120.0 1 2 3 5 −120.0 1 2 3 6 0.0
```

Description:

Restrain the motion of the specified atoms by making them part of a bond or angle or dihedral interaction whose strength can vary over time during a simulation. This is functionally equivalent to creating a bond or angle or dihedral for the atoms in a data file, as specified by the read_data command, albeit with a time−varying pre−factor coefficient. For the purpose of forcefield parameter−fitting or mapping a molecular potential energy surface, this fix reduces the hassle and risk associated with modifying data files. In other words, use this fix to temporarily force a molecule to adopt a particular conformation. To form a permanent bond or angle or dihedral, modify the data file.

The first example above applies a restraint to hold the dihedral angle formed by atoms 1, 2, 3, and 4 near 120 degrees using a constant restraint coefficient. The second example applies similar restraints to multiple dihedral angles using a restraint coefficient that increases from 0.0 to 2000.0 over the course of the run.

IMPORTANT NOTE: Adding a force to atoms implies a change in their potential energy as they move due to the applied force field. For dynamics via the run command, this energy can be added to the system's potential energy for thermodynamic output (see below). For energy minimization via the minimize command, this energy must be added to the system's potential energy to formulate a self−consistent minimization problem (see below).

In order for a restraint to be effective, the restraint force must typically be significantly larger than the forces associated with conventional forcefield terms. If the restraint is applied during a dynamics run (as opposed to during an energy minimization), a large restraint coefficient can significantly reduce the stable timestep size, especially if the atoms are initially far from the preferred conformation. You may need to experiment to determine what value of K works best for a given application.

For the case of finding a minimum energy structure for a single molecule with particular restratins (e.g. for fitting forcefield parameters or constructing a potential energy surface), commands such as the following might be useful:

```
# minimize molecule energy with restraints
velocity all create 600.0 8675309 mom yes rot yes dist gaussian
```

```
fix NVE all nve
fix TFIX all langevin 600.0 0.0 100 24601
fix REST all restrain 0.0 5000.0 dihedral 2 1 3 8 $angle1 3 1 2 9 $angle2
fix_modify REST energy yes
run 10000
fix TFIX all langevin 0.0 0.0 100 24601
fix REST all restrain 5000.0 5000.0 dihedral 2 1 3 8 $angle1 3 1 2 9 $angle2
fix_modify REST energy yes
run 10000
# sanity check for convergence
minimize 1e-6 1e-9 1000 100000
# report unrestrained energies
unfix REST
run 0
```

The *dihedral* keyword applies a dihedral restraint to the specified atoms using a simplified form of the function used in dihedral_style charmm. Specifically, the potential associated with the restraint is

$$E = K[1 + \cos(n\phi - d)]$$

with the following coefficients:

- K (energy) = K (specified above)
- n = 1
- d (degrees) = 180.0 + target (specified above)

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files.

The fix_modify *energy* option is supported by this fix to add the potential energy associated with this fix to the system's potential energy as part of thermodynamic output.

IMPORTANT NOTE: If you want the fictitious potential energy associated with the added forces to be included in the total potential energy of the system (the quantity being minimized), you MUST enable the fix_modify *energy* option for this fix.

This fix computes a global scalar, which can be accessed by various output commands. The scalar is the potential energy discussed above. The scalar value calculated by this fix is "extensive".

No parameter of this fix can be used with the *start/stop* keywords of the run command.

Restrictions:

The group−ID specified by this fix is ignored.

Currently, only dihedral restraints are allowed, but modification of the code to allow angle and bond restraints would be straightforward.

Related commands: none

Default: none

fix rigid command

fix rigid/nve command

fix rigid/nvt command

Syntax:

```
fix ID group−ID style bodystyle args keyword values ...
```

- ID, group−ID are documented in fix command
- style = *rigid* or *rigid/nve* or *rigid/nvt*
- bodystyle = *single* or *molecule* or *group*

```
  single args = none
    molecule args = none
    group args = N groupID1 groupID2 ...
      N = # of groups
      groupID1, groupID2, ... = list of N group IDs
```

- zero or more keyword/value pairs may be appended
- keyword = *langevin* or *temp* or *tparam* or *force* or *torque*

```
  langevin values = Tstart Tstop Tperiod seed
      Tstart,Tstop = desired temperature at start/stop of run (temperature units)
      Tdamp = temperature damping parameter (time units)
      seed = random number seed to use for white noise (positive integer)
    temp values = Tstart Tstop Tdamp
      Tstart,Tstop = desired temperature at start/stop of run (temperature units)
      Tdamp = temperature damping parameter (time units)
    tparam values = Tchain Titer Torder
      Tchain = length of Nose/Hoover thermostat chain
      Titer = number of thermostat iterations performed
      Torder = 3 or 5 = Yoshida−Suzuki integration parameters
    force values = M xflag yflag zflag
      M = which rigid body from 1−Nbody (see asterisk form below)
      xflag,yflag,zflag = off/on if component of center−of−mass force is active
    torque values = M xflag yflag zflag
      M = which rigid body from 1−Nbody (see asterisk form below)
      xflag,yflag,zflag = off/on if component of center−of−mass torque is active
```

Examples:

```
fix 1 clump rigid single
fix 1 clump rigid single force 1 off off on langevin 1.0 1.0 1.0 428984
fix 1 polychains rigid/nvt molecule temp 1.0 1.0 5.0
fix 1 polychains rigid molecule force 1*5 off off off force 6*10 off off on
fix 2 fluid rigid group 3 clump1 clump2 clump3 torque * off off off
```

Description:

Treat one or more sets of atoms as independent rigid bodies. This means that each timestep the total force and torque on each rigid body is computed as the sum of the forces and torques on its constituent particles and the coordinates, velocities, and orientations of the atoms in each body are updated so that the body moves and rotates as a single entity.

Examples of large rigid bodies are a large colloidal particle, or portions of a large biomolecule such as a protein.

Example of small rigid bodies are patchy nanoparticles, such as those modeled in this paper by Sharon Glotzer's group, clumps of granular particles, lipid molecules consiting of one or more point dipoles connected to other spheroids or ellipsoids, and coarse−grain models of nano or colloidal particles consisting of a small number of constituent particles. Note that the fix shake command can also be used to rigidify small molecules of 2, 3, or 4 atoms, e.g. water molecules. That fix treats the constituent atoms as point masses.

These fixes also update the positions and velocities of the atoms in each rigid body via time integration. The *rigid* and *rigid/nve* styles do this via constant NVE integration. The only difference is that the *rigid* style uses an integration technique based on Richardson iterations. The *rigid/nve* style uses the methods described in the paper by Miller, which are thought to provide better energy conservation than an iterative approach.

The *rigid/nvt* style performs constant NVT integration using a Nose/Hoover thermostat with chains as described originally in (Hoover) and (Martyna), which thermostats both the translational and rotational degrees of freedom of the rigid bodies. The rigid−body algorithm used by *rigid/nvt* is described in the paper by Kamberaj.

IMPORTANT NOTE: You should not update the atoms in rigid bodies via other time−integration fixes (e.g. nve, nvt, npt), or you will be integrating their motion more than once each timestep.

IMPORTANT NOTE: These fixes are overkill if you simply want to hold a collection of atoms stationary or have them move with a constant velocity. A simpler way to hold atoms stationary is to not include those atoms in your time integration fix. E.g. use "fix 1 mobile nve" instead of "fix 1 all nve", where "mobile" is the group of atoms that you want to move. You can move atoms with a constant velocity by assigning them an initial velocity (via the velocity command), setting the force on them to 0.0 (via the fix setforce command), and integrating them as usual (e.g. via the fix nve command).

The constituent particles within a rigid body can be point particles (the default in LAMMPS) or finite−size particles, such as spheres and ellipsoids. See the atom_style sphere and ellipsoid commands for more details on these kinds of particles. Finite−size particles contribute differently to the moment of inertia of a rigid body than do point particles. Finite−size particles can also experience torque (e.g. due to frictional granular interactions) and have an orientation. These contributions are accounted for by these fixes.

Forces between particles within a body do not contribute to the external force or torque on the body. Thus for computational efficiency, you may wish to turn off pairwise and bond interactions between particles within each rigid body. The neigh_modify exclude and delete_bonds commands are used to do this. For finite−size particles this also means the particles can be highly overlapped when creating the rigid body.

Each body must have two or more atoms. An atom can belong to at most one rigid body. Which atoms are in which bodies can be defined via several options.

For bodystyle *single* the entire fix group of atoms is treated as one rigid body.

For bodystyle *molecule*, each set of atoms in the fix group with a different molecule ID is treated as a rigid body.

For bodystyle *group*, each of the listed groups is treated as a separate rigid body. Only atoms that are also in the fix group are included in each rigid body.

IMPORTANT NOTE: To compute the initial center−of−mass position and other properties of each rigid body, the image flags for each atom in the body are used to "unwrap" the atom coordinates. Thus you must insure that these image flags are consistent so that the unwrapping creates a valid rigid body (one where the atoms are close together), particularly if the atoms in a single rigid body straddle a periodic boundary. This means the input data

file or restart file must define the image flags for each atom consistently or that you have used the set command to specify them correctly. If a dimension is non−periodic then the image flag of each atom must be 0 in that dimension, else an error is generated.

By default, each rigid body is acted on by other atoms which induce an external force and torque on its center of mass, causing it to translate and rotate. Components of the external center−of−mass force and torque can be turned off by the *force* and *torque* keywords. This may be useful if you wish a body to rotate but not translate, or vice versa, or if you wish it to rotate or translate continuously unaffected by interactions with other particles. Note that if you expect a rigid body not to move or rotate by using these keywords, you must insure its initial center−of−mass translational or angular velocity is 0.0. Otherwise the initial translational or angular momentum the body has will persist.

An xflag, yflag, or zflag set to *off* means turn off the component of force of torque in that dimension. A setting of *on* means turn on the component, which is the default. Which rigid body(s) the settings apply to is determined by the first argument of the *force* and *torque* keywords. It can be an integer M from 1 to Nbody, where Nbody is the number of rigid bodies defined. A wild−card asterisk can be used in place of, or in conjunction with, the M argument to set the flags for multiple rigid bodies. This takes the form "*" or "*n" or "n*" or "m*n". If N = the number of rigid bodies, then an asterisk with no numeric values means all bodies from 1 to N. A leading asterisk means all bodies from 1 to n (inclusive). A trailing asterisk means all bodies from n to N (inclusive). A middle asterisk means all types from m to n (inclusive). Note that you can use the *force* or *torque* keywords as many times as you like. If a particular rigid body has its component flags set multiple times, the settings from the final keyword are used.

For computational efficiency, you may wish to turn off pairwise and bond interactions within each rigid body, as they no longer contribute to the motion. The neigh_modify exclude and delete_bonds commands are used to do this.

For computational efficiency, you should typically define one fix rigid which includes all the desired rigid bodies. LAMMPS will allow multiple rigid fixes to be defined, but it is more expensive.

The keyword/value option pairs are used in the following ways.

The *langevin* and *temp* and *tparam* keywords perform thermostatting of the rigid bodies, altering both their translational and rotational degrees of freedom. What is meant by "temperature" of a collection of rigid bodies and how it can be monitored via the fix output is discussed below.

The *langevin* keyword applies a Langevin thermostat to the constant NVE time integration performed by either the *rigid* or *rigid/nve* styles. It cannot be used with the *rigid/nvt* style. The desired temperature at each timestep is a ramped value during the run from *Tstart* to *Tstop*. The *Tdamp* parameter is specified in time units and determines how rapidly the temperature is relaxed. For example, a value of 100.0 means to relax the temperature in a timespan of (roughly) 100 time units (tau or fmsec or psec − see the units command). The random # *seed* must be a positive integer. The way the Langevin thermostatting operates is explained on the fix langevin doc page.

The *temp* and *tparam* keywords apply a Nose/Hoover thermostat to the NVT time integration performed by the *rigid/nvt* style. They cannot be used with the *rigid* or *rigid/nve* styles. The desired temperature at each timestep is a ramped value during the run from *Tstart* to *Tstop*. The *Tdamp* parameter is specified in time units and determines how rapidly the temperature is relaxed. For example, a value of 100.0 means to relax the temperature in a timespan of (roughly) 100 time units (tau or fmsec or psec − see the units command).

Nose/Hoover chains are used in conjunction with this thermostat. The *tparam* keyword can optionally be used to change the chain settings used. *Tchain* is the number of thermostats in the Nose Hoover chain. This value, along

with *Tdamp* can be varied to dampen undesirable oscillations in temperature that can occur in a simulation. As a rule of thumb, increasing the chain length should lead to smaller oscillations.

IMPORTANT NOTE: There are alternate ways to thermostat a system of rigid bodies. You can use fix langevin to treat the individual particles in the rigid bodies as effectively immersed in an implicit solvent, e.g. a Brownian dynamics model. For hybrid systems with both rigid bodies and solvent particles, you can thermostat only the solvent particles that surround one or more rigid bodies by appropriate choice of groups in the compute and fix commands for temperature and thermostatting. The solvent interactions with the rigid bodies should then effectively thermostat the rigid body temperature as well without use of the Langevin or Nose/Hoover options associated with the fix rigid commands.

The keyword/value option pairs are used in the following ways.

If you use a temperature compute with a group that includes particles in rigid bodies, the degrees−of−freedom removed by each rigid body are accounted for in the temperature (and pressure) computation, but only if the temperature group includes all the particles in a particular rigid body.

A 3d rigid body has 6 degrees of freedom (3 translational, 3 rotational), except for a collection of point particles lying on a straight line, which has only 5, e.g a dimer. A 2d rigid body has 3 degrees of freedom (2 translational, 1 rotational).

IMPORTANT NOTE: You may wish to explicitly subtract additional degrees−of−freedom if you use the *force* and *torque* keywords to eliminate certain motions of one or more rigid bodies. LAMMPS does not do this automatically.

The rigid body contribution to the pressure of the system (virial) is also accounted for by this fix.

IMPORTANT NOTE: The periodic image flags of atoms in rigid bodies are altered so that the rigid body can be reconstructed correctly when it straddles periodic boundaries. The atom image flags are not incremented/decremented as they would be for non−rigid atoms as the rigid body crosses periodic boundaries. This means you cannot interpret them as you normally would. For example, the image flag values written to a dump file will be different than they would be if the atoms were not in a rigid body. Likewise the compute msd will not compute the expected mean−squared displacement for such atoms if the body moves across periodic boundaries. It also means that if you have bonds between a pair of rigid bodies and the bond straddles a periodic boundary, you cannot use the replicate command to increase the system size. Note that this fix does define image flags for each rigid body, which are incremented when the rigid body crosses a periodic boundary in the usual way. These image flags have the same meaning as atom images (see the "dump" command) and can be accessed and output as described below.

Restart, fix_modify, output, run start/stop, minimize info:

No information about the *rigid* and *rigid/nve* fixes are written to binary restart files. For style *rigid/nvt* the state of the Nose/Hoover thermostat is written to binary restart files. See the read_restart command for info on how to re−specify a fix in an input script that reads a restart file, so that the operation of the fix continues in an uninterrupted fashion.

The fix_modify *energy* option is supported by the rigid/nvt fix to add the energy change induced by the thermostatting to the system's potential energy as part of thermodynamic output.

The rigid and rigid/nve fixes computes a global scalar which can be accessed by various output commands. The scalar value calculated by these fixes is "intensive". The scalar is the current temperature of the collection of rigid bodies. This is averaged over all rigid bodies and their translational and rotational degrees of freedom. The

translational energy of a rigid body is 1/2 m v^2, where m = total mass of the body and v = the velocity of its center of mass. The rotational energy of a rigid body is 1/2 I w^2, where I = the moment of inertia tensor of the body and w = its angular velocity. Degrees of freedom constrained by the *force* and *torque* keywords are removed from this calculation.

The rigid/nvt fix computes a global scalar which can be accessed by various output commands. The scalar value calculated by the rigid/nvt fix is "extensive". The scalar is the cumulative energy change due to the thermostatting the fix performs.

All of these fixes compute a global array of values which can be accessed by various output commands. The number of rows in the array is equal to the number of rigid bodies. The number of columns is 15. Thus for each rigid body, 15 values are stored: the xyz coords of the center of mass (COM), the xyz components of the COM velocity, the xyz components of the force acting on the COM, the xyz components of the torque acting on the COM, and the xyz image flags of the COM, which have the same meaning as image flags for atom positions (see the "dump" command). The force and torque values in the array are not affected by the *force* and *torque* keywords in the fix rigid command; they reflect values before any changes are made by those keywords.

The ordering of the rigid bodies (by row in the array) is as follows. For the *single* keyword there is just one rigid body. For the *molecule* keyword, the bodies are ordered by ascending molecule ID. For the *group* keyword, the list of group IDs determines the ordering of bodies.

The array values calculated by these fixes are "intensive", meaning they are independent of the number of atoms in the simulation.

No parameter of these fixes can be used with the *start/stop* keywords of the run command. These fixes are not invoked during energy minimization.

Restrictions:

These fixes performs an MPI_Allreduce each timestep that is proportional in length to the number of rigid bodies. Hence they will not scale well in parallel if large numbers of rigid bodies are simulated.

Related commands:

delete_bonds, neigh_modify exclude

Default:

The option defaults are force * on on on and torque * on on on, meaning all rigid bodies are acted on by center−of−mass force and torque. Also Tchain = 10, Titer = 1, Torder = 3.

(Hoover) Hoover, Phys Rev A, 31, 1695 (1985).

(Kamberaj) Kamberaj, Low, Neal, J Chem Phys, 122, 224114 (2005).

(Martyna) Martyna, Klein, Tuckerman, J Chem Phys, 97, 2635 (1992); Martyna, Tuckerman, Tobias, Klein, Mol Phys, 87, 1117.

(Miller) Miller, Eleftheriou, Pattnaik, Ndirango, and Newns, J Chem Phys, 116, 8649 (2002).

(Zhang) Zhang, Glotzer, Nanoletters, 4, 1407−1413 (2004).

fix setforce command

fix setforce/cuda command

Syntax:

```
fix ID group-ID setforce fx fy fz keyword value ...
```

- ID, group−ID are documented in fix command
- setforce = style name of this fix command
- fx,fy,fz = force component values
- any of fx,fy,fz can be a variable (see below)
- zero or more keyword/value pairs may be appended to args
- keyword = *region*

```
region value = region-ID
    region-ID = ID of region atoms must be in to have added force
```

Examples:

```
fix freeze indenter setforce 0.0 0.0 0.0
fix 2 edge setforce NULL 0.0 0.0
fix 2 edge setforce NULL 0.0 v_oscillate
```

Description:

Set each component of force on each atom in the group to the specified values fx,fy,fz. This erases all previously computed forces on the atom, though additional fixes could add new forces. This command can be used to freeze certain atoms in the simulation by zeroing their force, either for running dynamics or performing an energy minimization. For dynamics, this assumes their initial velocity is also zero.

Any of the fx,fy,fz values can be specified as NULL which means do not alter the force component in that dimension.

Any of the 3 quantities defining the force components can be specified as an equal−style or atom−style variable, namely *fx*, *fy*, *fz*. If the value is a variable, it should be specified as v_name, where name is the variable name. In this case, the variable will be evaluated each timestep, and its value used to determine the force component.

Equal−style variables can specify formulas with various mathematical functions, and include thermo_style command keywords for the simulation box parameters and timestep and elapsed time. Thus it is easy to specify a time−dependent force field.

Atom−style variables can specify the same formulas as equal−style variables but can also include per−atom values, such as atom coordinates. Thus it is easy to specify a spatially−dependent force field with optional time−dependence as well.

If the *region* keyword is used, the atom must also be in the specified geometric region in order to have force added to it.

Styles with a *cuda* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in this section of the manual. The

accelerated styles take the same arguments and should produce the same results, except for round−off and precision issues.

These accelerated styles are part of the "user−cuda" package. They are only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the −suffix command−line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See this section of the manual for more instructions on how to use the accelerated styles effectively.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix.

This fix computes a global 3−vector of forces, which can be accessed by various output commands. This is the total force on the group of atoms before the forces on individual atoms are changed by the fix. The vector values calculated by this fix are "extensive".

No parameter of this fix can be used with the *start/stop* keywords of the run command.

The forces due to this fix are imposed during an energy minimization, invoked by the minimize command, but you cannot set forces to any value besides zero when performing a minimization. Use the fix addforce command if you want to apply a non−zero force to atoms during a minimization.

Restrictions: none

Related commands:

fix addforce, fix aveforce

Default: none

fix shake command

fix shake/cuda command

Syntax:

```
fix ID group-ID shake tol iter N keyword values ...
```

- ID, group−ID are documented in fix command
- shake = style name of this fix command
- tol = accuracy tolerance of SHAKE solution
- iter = max # of iterations in each SHAKE solution
- N = print SHAKE statistics every this many timesteps (0 = never)
- one or more keyword/value pairs are appended
- keyword = *b* or *a* or *t* or *m*

```
  b values = one or more bond types
    a values = one or more angle types
    t values = one or more atom types
    m value = one or more mass values
```

Examples:

```
fix 1 sub shake 0.0001 20 10 b 4 19 a 3 5 2
fix 1 sub shake 0.0001 20 10 t 5 6 m 1.0 a 31
```

Description:

Apply bond and angle constraints to specified bonds and angles in the simulation. This typically enables a longer timestep.

Each timestep the specified bonds and angles are reset to their equilibrium lengths and angular values via the well−known SHAKE algorithm. This is done by applying an additional constraint force so that the new positions preserve the desired atom separations. The equations for the additional force are solved via an iterative method that typically converges to an accurate solution in a few iterations. The desired tolerance (e.g. 1.0e−4 = 1 part in 10000) and maximum # of iterations are specified as arguments. Setting the N argument will print statistics to the screen and log file about regarding the lengths of bonds and angles that are being constrained. Small delta values mean SHAKE is doing a good job.

In LAMMPS, only small clusters of atoms can be constrained. This is so the constraint calculation for a cluster can be performed by a single processor, to enable good parallel performance. A cluster is defined as a central atom connected to others in the cluster by constrained bonds. LAMMPS allows for the following kinds of clusters to be constrained: one central atom bonded to 1 or 2 or 3 atoms, or one central atom bonded to 2 others and the angle between the 3 atoms also constrained. This means water molecules or CH2 or CH3 groups may be constrained, but not all the C−C backbone bonds of a long polymer chain.

The *b* keyword lists bond types that will be constrained. The *t* keyword lists atom types. All bonds connected to an atom of the specified type will be constrained. The *m* keyword lists atom masses. All bonds connected to atoms of the specified masses will be constrained (within a fudge factor of MASSDELTA specified in fix_shake.cpp). The *a* keyword lists angle types. If both bonds in the angle are constrained then the angle will also be constrained if its type is in the list.

For all keywords, a particular bond is only constrained if both atoms in the bond are in the group specified with the SHAKE fix.

The degrees−of−freedom removed by SHAKE bonds and angles are accounted for in temperature and pressure computations. Similarly, the SHAKE contribution to the pressure of the system (virial) is also accounted for.

Styles with a *cuda* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in this section of the manual. The accelerated styles take the same arguments and should produce the same results, except for round−off and precision issues.

These accelerated styles are part of the "user−cuda" package. They are only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the −suffix command−line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See this section of the manual for more instructions on how to use the accelerated styles effectively.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix. No global or per−atom quantities are stored by this fix for access by various output commands. No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions:

For computational efficiency, there can only be one shake fix defined in a simulation.

If you use a tolerance that is too large or a max−iteration count that is too small, the constraints will not be enforced very strongly, which can lead to poor energy conservation. You can test for this in your system by running a constant NVE simulation with a particular set of SHAKE parameters and monitoring the energy versus time.

Related commands: none

Default: none

fix smd command

Syntax:

```
fix ID group-ID smd type values keyword values
```

- ID, group−ID are documented in fix command
- smd = style name of this fix command
- mode = *cvel* or *cfor* to select constant velocity or constant force SMD

```
  cvel values = K vel
      K = spring constant (force/distance units)
      vel = velocity of pulling (distance/time units)
    cfor values = force
      force = pulling force (force units)
```

- keyword = *tether* or *couple*

```
  tether values = x y z R0
      x,y,z = point to which spring is tethered
      R0 = distance of end of spring from tether point (distance units)
    couple values = group-ID2 x y z R0
      group-ID2 = 2nd group to couple to fix group with a spring
      x,y,z = direction of spring, automatically computed with 'auto'
      R0 = distance of end of spring (distance units)
```

Examples:

```
fix  pull    cterm smd cvel 20.0 -0.00005 tether NULL NULL 100.0 0.0
fix  pull    cterm smd cvel 20.0 -0.0001 tether 25.0 25 25.0 0.0
fix  stretch cterm smd cvel 20.0  0.0001 couple nterm auto auto auto 0.0
fix  pull    cterm smd cfor  5.0 tether 25.0 25.0 25.0 0.0
```

Description:

This fix implements several options of steered MD (SMD) as reviewed in (Izrailev), which allows to induce conformational changes in systems and to compute the potential of mean force (PMF) along the assumed reaction coordinate (Park) based on Jarzynski's equality (Jarzynski). This fix borrows a lot from fix spring and fix setforce.

You can apply a moving spring force to a group of atoms (*tether* style) or between two groups of atoms (*couple* style). The spring can then be used in either constant velocity (*cvel*) mode or in constant force (*cfor*) mode to induce transitions in your systems. When running in *tether* style, you may need some way to fix some other part of the system (e.g. via fix spring/self)

The *tether* style attaches a spring between a point at a distance of R0 away from a fixed point *x*,*y*,*z* and the center of mass of the fix group of atoms. A restoring force of magnitude K (R − R0) Mi / M is applied to each atom in the group where *K* is the spring constant, Mi is the mass of the atom, and M is the total mass of all atoms in the group. Note that *K* thus represents the total force on the group of atoms, not a per−atom force.

In *cvel* mode the distance R is incremented or decremented monotonously according to the pulling (or pushing) velocity. In *cfor* mode a constant force is added and the actual distance in direction of the spring is recorded.

The *couple* style links two groups of atoms together. The first group is the fix group; the second is specified by group−ID2. The groups are coupled together by a spring that is at equilibrium when the two groups are displaced

by a vector in direction *x*,*y*,*z* with respect to each other and at a distance R0 from that displacement. Note that *x*,*y*,*z* only provides a direction and will be internally normalized. But since it represents the *absolute* displacement of group−ID2 relative to the fix group, (1,1,0) is a different spring than (−1,−1,0). For each vector component, the displacement can be described with the *auto* parameter. In this case the direction is recomputed in every step, which can be useful for steering a local process where the whole object undergoes some other change. When the relative positions and distance between the two groups are not in equilibrium, the same spring force described above is applied to atoms in each of the two groups.

For both the *tether* and *couple* styles, any of the x,y,z values can be specified as NULL which means do not include that dimension in the distance calculation or force application.

For constant velocity pulling (*cvel* mode), the running integral over the pulling force in direction of the spring is recorded and can then later be used to compute the potential of mean force (PMF) by averaging over multiple independent trajectories along the same pulling path.

Restart, fix_modify, output, run start/stop, minimize info:

The fix stores the direction of the spring, current pulling target distance and the running PMF to binary restart files. See the read_restart command for info on how to re−specify a fix in an input script that reads a restart file, so that the operation of the fix continues in an uninterrupted fashion.

None of the fix_modify options are relevant to this fix.

This fix computes a vector list of 7 quantities, which can be accessed by various output commands. The quantities in the vector are in this order: the x−, y−, and z−component of the pulling force, the total force in direction of the pull, the equilibrium distance of the spring, the distance between the two reference points, and finally the accumulated PMF (the sum of pulling forces times displacement).

The force is the total force on the group of atoms by the spring. In the case of the *couple* style, it is the force on the fix group (group−ID) or the negative of the force on the 2nd group (group−ID2). The vector values calculated by this fix are "extensive".

No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions:

This fix is part of the "user−misc" package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

Related commands:

fix drag, fix spring, fix spring/self, fix spring/rg

Default: none

(Izrailev) Izrailev, Stepaniants, Isralewitz, Kosztin, Lu, Molnar, Wriggers, Schulten. Computational Molecular Dynamics: Challenges, Methods, Ideas, volume 4 of Lecture Notes in Computational Science and Engineering, pp. 39−65. Springer−Verlag, Berlin, 1998.

(Park) Park, Schulten, J. Chem. Phys. 120 (13), 5946 (2004)

(Jarzynski) Jarzynski, Phys. Rev. Lett. 78, 2690 (1997)

fix spring command

Syntax:

```
fix ID group−ID spring keyword values
```

- ID, group−ID are documented in fix command
- spring = style name of this fix command
- keyword = *tether* or *couple*

```
  tether values = K x y z R0
      K = spring constant (force/distance units)
      x,y,z = point to which spring is tethered
      R0 = equilibrium distance from tether point (distance units)
    couple values = group−ID2 K x y z R0
      group−ID2 = 2nd group to couple to fix group with a spring
      K = spring constant (force/distance units)
      x,y,z = direction of spring
      R0 = equilibrium distance of spring (distance units)
```

Examples:

```
fix pull ligand spring tether 50.0 0.0 0.0 0.0 0.0
fix pull ligand spring tether 50.0 0.0 0.0 0.0 5.0
fix pull ligand spring tether 50.0 NULL NULL 2.0 3.0
fix 5 bilayer1 spring couple bilayer2 100.0 NULL NULL 10.0 0.0
fix longitudinal pore spring couple ion 100.0 NULL NULL −20.0 0.0
fix radial pore spring couple ion 100.0 0.0 0.0 NULL 5.0
```

Description:

Apply a spring force to a group of atoms or between two groups of atoms. This is useful for applying an umbrella force to a small molecule or lightly tethering a large group of atoms (e.g. all the solvent or a large molecule) to the center of the simulation box so that it doesn't wander away over the course of a long simulation. It can also be used to hold the centers of mass of two groups of atoms at a given distance or orientation with respect to each other.

The *tether* style attaches a spring between a fixed point *x*,*y*,*z* and the center of mass of the fix group of atoms. The equilibrium position of the spring is R0. At each timestep the distance R from the center of mass of the group of atoms to the tethering point is computed, taking account of wrap−around in a periodic simulation box. A restoring force of magnitude K (R − R0) Mi / M is applied to each atom in the group where *K* is the spring constant, Mi is the mass of the atom, and M is the total mass of all atoms in the group. Note that *K* thus represents the total force on the group of atoms, not a per−atom force.

The *couple* style links two groups of atoms together. The first group is the fix group; the second is specified by group−ID2. The groups are coupled together by a spring that is at equilibrium when the two groups are displaced by a vector *x*,*y*,*z* with respect to each other and at a distance R0 from that displacement. Note that *x*,*y*,*z* is the equilibrium displacement of group−ID2 relative to the fix group. Thus (1,1,0) is a different spring than (−1,−1,0). When the relative positions and distance between the two groups are not in equilibrium, the same spring force described above is applied to atoms in each of the two groups.

For both the *tether* and *couple* styles, any of the x,y,z values can be specified as NULL which means do not include that dimension in the distance calculation or force application.

The first example above pulls the ligand towards the point (0,0,0). The second example holds the ligand near the surface of a sphere of radius 5 around the point (0,0,0). The third example holds the ligand a distance 3 away from the z=2 plane (on either side).

The fourth example holds 2 bilayers a distance 10 apart in z. For the last two examples, imagine a pore (a slab of atoms with a cylindrical hole cut out) oriented with the pore axis along z, and an ion moving within the pore. The fifth example holds the ion a distance of −20 below the z = 0 center plane of the pore (umbrella sampling). The last example holds the ion a distance 5 away from the pore axis (assuming the center−of−mass of the pore in x,y is the pore axis).

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files.

The fix_modify *energy* option is supported by this fix to add the energy stored in the spring to the system's potential energy as part of thermodynamic output.

This fix computes a global scalar which can be accessed by various output commands. The scalar is the spring energy = 0.5 * K * r^2.

This fix also computes global 4−vector which can be accessed by various output commands. The first 3 quantities in the vector are xyz components of the total force added to the group of atoms by the spring. In the case of the *couple* style, it is the force on the fix group (group−ID) or the negative of the force on the 2nd group (group−ID2). The 4th quantity in the vector is the magnitude of the force added by the spring, as a positive value if (r−R0) > 0 and a negative value if (r−R0) < 0. This sign convention can be useful when using the spring force to compute a potential of mean force (PMF).

The scalar and vector values calculated by this fix are "extensive".

No parameter of this fix can be used with the *start/stop* keywords of the run command.

The forces due to this fix are imposed during an energy minimization, invoked by the minimize command.

IMPORTANT NOTE: If you want the spring energy to be included in the total potential energy of the system (the quantity being minimized), you MUST enable the fix_modify *energy* option for this fix.

Restrictions: none

Related commands:

fix drag, fix spring/self, fix spring/rg, fix smd

Default: none

fix spring/rg command

Syntax:

```
fix ID group-ID spring/rg K RG0
```

- ID, group−ID are documented in fix command
- spring/rg = style name of this fix command
- K = harmonic force constant (force/distance units)
- RG0 = target radius of gyration to constrain to (distance units)

```
  if RG0 = NULL, use the current RG as the target value
```

Examples:

```
fix 1 protein spring/rg 5.0 10.0
fix 2 micelle spring/rg 5.0 NULL
```

Description:

Apply a harmonic restraining force to atoms in the group to affect their central moment about the center of mass (radius of gyration). This fix is useful to encourage a protein or polymer to fold/unfold and also when sampling along the radius of gyration as a reaction coordinate (i.e. for protein folding).

The radius of gyration is defined as RG in the first formula. The energy of the constraint and associated force on each atom is given by the second and third formulas, when the group is at a different RG than the target value RG0.

$$R_G{}^2 = \frac{1}{M}\sum_i^N m_i \left(x_i - \frac{1}{M}\sum_j^N m_j x_j\right)^2$$

$$E = K\left(R_G - R_{G0}\right)^2$$

$$F_i = 2K\frac{m_i}{M}\left(1 - \frac{R_{G0}}{R_G}\right)\left(x_i - \frac{1}{M}\sum_j^N m_j x_j\right)$$

The (xi − center−of−mass) term is computed taking into account periodic boundary conditions, m_i is the mass of the atom, and M is the mass of the entire group. Note that K is thus a force constant for the aggregate force on the group of atoms, not a per−atom force.

If RG0 is specified as NULL, then the RG of the group is computed at the time the fix is specified, and that value is used as the target.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix. No global or per−atom quantities are stored by this fix for access by various output commands. No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy

minimization.

Restrictions: none

Related commands:

fix spring, fix spring/self fix drag, fix smd

Default: none

fix spring/self command

Syntax:

```
fix ID group-ID spring/self K
```

- ID, group−ID are documented in fix command
- spring/self = style name of this fix command
- K = spring constant (force/distance units)

Examples:

```
fix tether boundary-atoms spring/self 10.0
```

Description:

Apply a spring force independently to each atom in the group to tether it to its initial position. The initial position for each atom is its location at the time the fix command was issued. At each timestep, the magnitude of the force on each atom is −Kr, where r is the displacement of the atom from its current position to its initial position.

Restart, fix_modify, output, run start/stop, minimize info:

This fix writes the original coordinates of tethered atoms to binary restart files, so that the spring effect will be the same in a restarted simulation. See the read_restart command for info on how to re−specify a fix in an input script that reads a restart file, so that the operation of the fix continues in an uninterrupted fashion.

The fix_modify *energy* option is supported by this fix to add the energy stored in the per−atom springs to the system's potential energy as part of thermodynamic output.

This fix computes a global scalar which can be accessed by various output commands. The scalar is an energy which is the sum of the spring energy for each atom, where the per−atom energy is 0.5 * K * r^2. The scalar value calculated by this fix is "extensive".

No parameter of this fix can be used with the *start/stop* keywords of the run command.

The forces due to this fix are imposed during an energy minimization, invoked by the minimize command.

IMPORTANT NOTE: If you want the per−atom spring energy to be included in the total potential energy of the system (the quantity being minimized), you MUST enable the fix_modify *energy* option for this fix.

Restrictions: none

Related commands:

fix drag, fix spring, fix smd, fix spring/rg

Default: none

fix srd command

Syntax:

```
fix ID group-ID srd N groupbig-ID Tsrd hgrid seed keyword value ...
```

- ID, group−ID are documented in fix command
- srd = style name of this fix command
- N = reset SRD particle velocities every this many timesteps
- groupbig−ID = ID of group of large particles that SRDs interact with
- Tsrd = temperature of SRD particles (temperature units)
- hgrid = grid spacing for SRD grouping (distance units)
- seed = random # seed (positive integer)

- zero or more keyword/value pairs may be appended
- keyword = *lamda* or *collision* or *overlap* or *inside* or *exact* or *radius* or *bounce* or *search* or *cubic* or *shift* or *stream*

```
lamda value = mean free path of SRD particles (distance units)
  collision value = noslip or slip = collision model
  overlap value = yes or no = whether big particles may overlap
  inside value = error or warn or ignore = how SRD particles which end up inside a big partic
  exact value = yes or no
  radius value = rfactor = scale collision radius by this factor
  bounce value = Nbounce = max # of collisions an SRD particle can undergo in one timestep
  search value = sgrid = grid spacing for collision partner searching (distance units)
  cubic values = style tolerance
    style = error or warn
    tolerance = fractional difference allowed (0 <= tol <= 1)
  shift values = style seed
    style = no or yes or possible
    seed = random # seed (positive integer)
  stream value = yes or no = whether or not streaming velocity is added for shear deformation
```

Examples:

```
fix 1 srd srd 10 big 1.0 0.25 482984
fix 1 srd srd 10 big 0.5 0.25 482984 collision slip search 0.5
```

Description:

Treat a group of partilces as stochastic rotation dynamics (SRD) particles that serve as a background solvent when interacting with big (colloidal) particles in groupbig−ID. The SRD formalism is described in (Hecht). The key idea behind using SRD particles as a cheap coarse−grained solvent is that SRD particles do not interact with each other, but only with the solute particles, which in LAMMPS can be spheroids, ellipsoids, or rigid bodies containing multiples spherioids and ellipsoids. The collision and rotation properties of the model imbue the SRD particles with fluid−like properties, including an effective viscosity. Thus simulations with large solute particles can be run more quickly, to measure solute propoerties like diffusivity and viscosity in a background fluid. The usual LAMMPS fixes for such simulations, such as fix deform, fix viscosity, and fix nvt/sllod, can be used in conjunction with the SRD model.

For more details on how the SRD model is implemented in LAMMPS, this paper describes the implementation and usage of pure SRD fluids. This paper, which is nearly complete, describes the implementation and usage of

mixture systems (solute particles in an SRD fluid). See the examples/srd directory for sample input scripts using SRD particles in both settings.

This fix does 2 things:

(1) It advects the SRD particles, performing collisions between SRD and big particles or walls every timestep, imparting force and torque to the big particles. Collisions also change the position and velocity of SRD particles.

(2) It resets the velocity distribution of SRD particles via random rotations every N timesteps.

SRD particles have a mass, temperature, characteristic timestep dt_SRD, and mean free path between collisions (lamda). The fundamental equation relating these 4 quantities is

```
lamda = dt_SRD * sqrt(Kboltz * Tsrd / mass)
```

The mass of SRD particles is set by the mass command elsewhere in the input script. The SRD timestep dt_SRD is N times the step dt defined by the timestep command. Big particles move in the normal way via a time integration fix with a short timestep dt. SRD particles advect with a large timestep dt_SRD >= dt.

If the *lamda* keyword is not specified, the the SRD temperature *Tsrd* is used in the above formula to compute lamda. If the *lamda* keyword is specified, then the *Tsrd* setting is ignored and the above equation is used to compute the SRD temperature.

The characteristic length scale for the SRD fluid is set by *hgrid* which is used to bin SRD particles for purposes of resetting their velocities. Normally hgrid is set to be 1/4 of the big particle diameter or smaller, to adequately resolve fluid properties around the big particles.

Lamda cannot be smaller than 0.6 * hgrid, else an error is generated (unless the *shift* keyword is used, see below). The velocities of SRD particles are bounded by Vmax, which is set so that an SRD particle will not advect further than Dmax = 4*lamda in dt_SRD. This means that roughly speaking, Dmax should not be larger than a big particle diameter, else SRDs may pass thru big particles without colliding. A warning is generated if this is the case.

Collisions between SRD particles and big particles or walls are modeled as a lightweight SRD point particle hitting a heavy big particle of given diameter or a wall at a point on its surface and bouncing off with a new velocity. The collision changes the momentum of the SRD particle. It imparts a force and torque to the big particle. It imparts a force to a wall. Static or moving SRD walls are setup via the fix wall/srd command. For the remainder of this doc page, a collision of an SRD particle with a wall can be viewed as a collision with a big particle of infinite radius and mass.

The *collision* keyword sets the style of collisions. The *slip* style means that the tangential component of the SRD particle momentum is preserved. Thus a force is imparted to a big particle, but no torque. The normal component of the new SRD velocity is sampled from a Gaussian distribution at temperature *Tsrd*.

For the *noslip* style, both the normal and tangential components of the new SRD velocity are sampled from a Gaussian distribution at temperature *Tsrd*. Additionally, a new tangential direction for the SRD velocity is chosen randomly. This collision style imparts torque to a big particle. Thus a time integrator fix that rotates the big particles appropriately should be used.

The *overlap* keyword should be set to *yes* if two (or more) big particles can ever overlap. This depends on the pair potential interaction used for big−big interactions, or could be the case if multiple big particles are held together as rigid bodies via the fix rigid command. If the *overlap* keyword is *no* and big particles do in fact overlap, then

SRD/big collisions can generate an error if an SRD ends up inside two (or more) big particles at once. How this error is treated is determined by the *inside* keyword. Running with *overlap* set to *no* allows for faster collision checking, so it should only be set to *yes* if needed.

The *inside* keyword determines how a collision is treated if the computation determines that the timestep started with the SRD particle already inside a big particle. If the setting is *error* then this generates an error message and LAMMPS stops. If the setting is *warn* then this generates a warning message and the code continues. If the setting is *ignore* then no message is generated. One of the output quantities logged by the fix (see below) tallies the number of such events, so it can be monitored. Note that once an SRD particle is inside a big particle, it may remain there for several steps until it drifts outside the big particle.

The *exact* keyword determines how accurately collisions are computed. A setting of *yes* computes the time and position of each collision as SRD and big particles move together. A setting of *no* estimates the position of each collision based on the end−of−timestep positions of the SRD and big particle. If *overlap* is set to yes, the setting of the *exact* keyword is ignored since time−accurate collisions are needed.

The *radius* keyword scales the effective size of big particles. If big particles will overlap as they undergo dynamics, then this keyword can be used to scale down their effective collision radius by an amount *rfactor*, so that SRD particle will only collide with one big particle at a time. For example, in a Lennard−Jones system at a temperature of 1.0 (in reduced LJ units), the minimum separation bewteen two big particles is as small as about 0.88 sigma. Thus an *rfactor* value of 0.85 should prevent dual collisions.

The *bounce* keyword can be used to limit the maximum number of collisions an SRD particle undergoes in a single timestep as it bounces between nearby big particles. Note that if the limit is reached, the SRD can be left inside a big particle. A setting of 0 is the same as no limit.

There are 2 kinds of bins created and maintained when running an SRD simulation. The first are "SRD bins" which are used to bin SRD particles and reset their velocities, as discussed above. The second are "search bins" which are used to identify SRD/big particle collisions.

The *search* keyword can be used to choose a search bin size for identifying SRD/big particle collisions. The default is to use the *hgrid* parameter for SRD bins as the search bin size. Choosing a smaller or large value may be more efficient, depending on the problem. But, in a statistical sense, it should not change the simulation results.

The *cubic* keyword can be used to generate an error or warning when the bin size chosen by LAMMPS creates SRD bins that are non−cubic or different than the requested value of *hgrid* by a specified *tolerance*. Note that using non−cubic SRD bins can lead to undetermined behavior when rotating the velocities of SRD particles, hence LAMMPS tries to protect you from this problem.

LAMMPS attempts to set the SRD bin size to exactly *hgrid*. However, there must be an integer number of bins in each dimension of the simulation box. Thus the actual bin size will depend on the size and shape of the overall simulation box. The actual bin size is printed as part of the SRD output when a simulation begins.

If the actual bin size in non−cubic by an amount exceeding the tolerance, an error or warning is printed, depending on the style of the *cubic* keyword. Likewise, if the actual bin size differs from the requested *hgrid* value by an amount exceeding the tolerance, then an error or warning is printed. The *tolerance* is a fractional difference. E.g. a tolerance setting of 0.01 on the shape means that if the ratio of any 2 bin dimensions exceeds (1 +/− tolerance) then an error or warning is generated. Similarly, if the ratio of any bin dimension with *hgrid* exceeds (1 +/− tolerance), then an error or warning is generated.

IMPORTANT NOTE: The fix srd command can be used with simluations the size and/or shape of the simulation box changes. This can be due to non−periodic boundary conditions or the use of fixes such as the fix deform or

fix wall/srd commands to impose a shear on an SRD fluid or an interaction with an external wall. If the box size changes then the size of SRD bins must be recalculated every reneighboring. This is not necessary if only the box shape changes. This re−binning is always done so as to fit an integer number of bins in the current box dimension, whether it be a fixed, shrink−wrapped, or periodic boundary, as set by the boundary command. If the box size or shape changes, then the size of the search bins must be recalculated avery reneighboring. Note that changing the SRD bin size may alter the properties of the SRD fluid, such as its viscosity.

The *shift* keyword determines whether the coordinates of SRD particles are randomly shifted when binned for purposes of rotating their velocities. When no shifting is performed, SRD particles are binned and the velocity distribution of the set of SRD particles in each bin is adjusted via a rotation operator. This is a statistically valid operation if SRD particles move sufficiently far between successive rotations. This is determined by their mean−free path lamda. If lamda is less than 0.6 of the SRD bin size, then shifting is required. A shift means that all of the SRD particles are shifted by a vector whose coordinates are chosen randomly in the range [−1/2 bin size, 1/2 bin size]. Note that all particles are shifted by the same vector. The specified random number seed is used to generate these vectors. This operation sufficiently randomizes which SRD particles are in the same bin, even if lamda is small.

If the *shift* style is set to *no*, then no shifting is performed, but bin data will be communicated if bins overlap processor boundaries. An error will be generated if lamda < 0.6 of the SRD bin size. If the *shift* style is set to *possible*, then shifting is performed only if lamda < 0.6 of the SRD bin size. A warning is generated to let you know this is occurring. If the *shift* style is set to *yes* then shifting is performed regardless of the magnitude of lamda.

The shift seed is not used if the *shift* style is set to *no*, but must still be specified.

Note that shifting of SRD coordinates requires extra communication, hence it should not normally be enabled unless required.

The *stream* keyword should be used when SRD particles are used with the fix deform command to perform a simulation undergoing shear, e.g. to measure a viscosity. If the *stream* style is set to *yes*, then the mean velocity of each bin of SRD particles is set to the streaming velocity of the deforming box, each time SRD velocities are reset, every N timesteps. If the *stream* style is set to *no*, then the mean velocity is unchanged, which may mean that it takes a long time for the SRD fluid to come to equilibrium with a velocity profile that matches the simulation box deformation.

IMPORTANT NOTE: This fix is normally used for simulations with a huge number of SRD particles relative to the number of big particles, e.g. 100 to 1. In this scenario, computations that involve only big particles (neighbor list creation, communication, time integration) can slow down dramatically due to the large number of background SRD particles.

Three other input script commands will largely overcome this effect, speeding up an SRD simulation by a significant amount. These are the atom_modify first, neigh_modify include, and communicate group commands. Each takes a group−ID as an argument, which in this case should be the group−ID of the big solute particles.

Additionally, when a pair_style for big/big particle interactions is specified, the pair_coeff command should be used to turn off big/SRD interactions, e.g. by setting their epsilon or cutoff length to 0.0.

The "delete_atoms overlap" command may be useful in setting up an SRD simulation to insure there are no initial overlaps between big and SRD particles.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix.

This fix tabulates several SRD statistics which are stored in a vector of length 12, which can be accessed by various output commands. The vector values calculated by this fix are "intensive", meaning they do not scale with the size of the simulation. Technically, the first 8 do scale with the size of the simulation, but treating them as intensive means they are not scaled when printed as part of thermodyanmic output.

These are the 12 quantities. All are values for the current timestep, except the last three which are cummulative quantities since the beginning of the run.

- (1) # of SRD/big collision checks performed
- (2) # of SRDs which had a collision
- (3) # of SRD/big colllisions (including multiple bounces)
- (4) # of SRD particles inside a big particle
- (5) # of SRD particles whose velocity was rescaled to be < Vmax
- (6) # of bins for collision searching
- (7) # of bins for SRD velocity rotation
- (8) # of bins in which SRD temperature was computed
- (9) SRD temperature
- (10) # of SRD particles which have undergone max # of bounces
- (11) max # of bounces any SRD particle has had in a single step
- (12) # of reneighborings dues to SRD particles moving too far

No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions:

This command can only be used if LAMMPS was built with the "srd" package. See the Making LAMMPS section for more info on packages.

Related commands:

fix wall/srd

Default:

The option defaults are lamda inferred from Tsrd, collision = noslip, overlap = no, inside = error, exact = yes, radius = 1.0, bounce = 0, search = hgrid, cubic = error 0.01, shift = no, stream = yes.

(Hecht) Hecht, Harting, Ihle, Herrmann, Phys Rev E, 72, 011408 (2005).

(Petersen) Petersen, Lechman, Plimpton, Grest, in' t Veld, Schunk, J Chem Phys, 132, 174106 (2010).

(Lechman) Lechman, et al, in preparation (2010).

fix store/force command

Syntax:

```
fix ID group-ID store/force
```

- ID, group−ID are documented in fix command
- store/force = style name of this fix command

Examples:

```
fix 1 all store/force
```

Description:

Store the forces on atoms in the group at the point during each timestep when the fix is invoked, as described below. This is useful for storing forces before constraints or other boundary conditions are computed which modify the forces, so that unmodified forces can be written to a dump file or accessed by other output commands that use per−atom quantities.

This fix is invoked at the point in the velocity−Verlet timestepping immediately after pair, bond, angle, dihedral, improper, and long−range forces have been calculated. It is the point in the timestep when various fixes that compute constraint forces are calculated and potentially modify the force on each atom. Examples of such fixes are fix shake, fix wall, and fix indent.

IMPORTANT NOTE: The order in which various fixes are applied which operate at the same point during the timestep, is the same as the order they are specified in the input script. Thus normally, if you want to store per−atom forces due to force field interactions, before constraints are applied, you should list this fix first within that set of fixes, i.e. before other fixes that apply constraints. However, if you wish to include certain constraints (e.g. fix shake) in the stored force, then it could be specified after some fixes and before others.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix.

This fix produces a per−atom array which can be accessed by various output commands. The number of columns for each atom is 3, and the columns store the x,y,z forces on each atom. The per−atom values be accessed on any timestep.

No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions: none

Related commands:

fix store_state

Default: none

fix store/state command

Syntax:

```
fix ID group−ID store/state N input1 input2 ... keyword value ...
```

- ID, group−ID are documented in fix command
- store/state = style name of this fix command
- N = store atom attributes every N steps, N = 0 for initial store only
- input = one or more atom attributes

```
possible attributes = id, mol, type, mass,
                      x, y, z, xs, ys, zs, xu, yu, zu, ix, iy, iz,
                      vx, vy, vz, fx, fy, fz,
                      q, mux, muy, muz,
                      radius, omegax, omegay, omegaz,
                      angmomx, angmomy, angmomz, tqx, tqy, tqz
                      c_ID, c_ID[N], f_ID, f_ID[N], v_name

    id = atom ID
    mol = molecule ID
    type = atom type
    mass = atom mass
    x,y,z = unscaled atom coordinates
    xs,ys,zs = scaled atom coordinates
    xu,yu,zu = unwrapped atom coordinates
    ix,iy,iz = box image that the atom is in
    vx,vy,vz = atom velocities
    fx,fy,fz = forces on atoms
    q = atom charge
    mux,muy,muz = orientation of dipolar atom
    radius = radius of extended spherical particle
    omegax,omegay,omegaz = angular velocity of extended particle
    angmomx,angmomy,angmomz = angular momentum of extended particle
    tqx,tqy,tqz = torque on extended particles
    c_ID = per−atom vector calculated by a compute with ID
    c_ID[I] = Ith column of per−atom array calculated by a compute with ID
    f_ID = per−atom vector calculated by a fix with ID
    f_ID[I] = Ith column of per−atom array calculated by a fix with ID
    v_name = per−atom vector calculated by an atom−style variable with name
```

- zero or more keyword/value pairs may be appended
- keyword = *com*

```
com value = yes or no
```

Examples:

```
fix 1 all store/state 0 x y z
fix 1 all store/state 0 xu yu zu com yes
fix 2 all store/state 1000 vx vy vz
```

Description:

Define a fix that stores attributes for each atom in the group at the time the fix is defined. If *N* is 0, then the values are never updated, so this is a way of archiving an atom attribute at a given time for future use in a calculation or output. See the discussion of output commands that take fixes as inputs. And see for example, the compute

reduce, fix ave/atom, fix ave/histo, fix ave/spatial, and atom−style variable commands.

If *N* is not zero, then the attributes will be updated every *N* steps.

IMPORTANT NOTE: Actually, only atom attributes specified by keywords like *xu* or *vy* are initially stored immediately at the point in your input script when the fix is defined. Attributes specified by a compute, fix, or variable are not initially stored until the first run following the fix definition begins. This is because calculating those attributes may require quantities that are not defined in between runs.

The list of possible attributes is the same as that used by the dump custom command, which describes their meaning.

If the *com* keyword is set to *yes* then the *xu*, *yu*, and *zu* inputs store the position of each atom relative to the center−of−mass of the group of atoms, instead of storing the absolute position. This option is used by the compute msd command.

The requested values are stored in a per−atom vector or array as discussed below. Zeroes are stored for atoms not in the specified group.

Restart, fix_modify, output, run start/stop, minimize info:

This fix writes the per−atom values it stores to binary restart files, so that the values can be restored when a simulation is restarted. See the read_restart command for info on how to re−specify a fix in an input script that reads a restart file, so that the operation of the fix continues in an uninterrupted fashion.

None of the fix_modify options are relevant to this fix.

If a single input is specified, this fix produces a per−atom vector. If multiple inputs are specified, a per−atom array is produced where the number of columns for each atom is the number of inputs. These can be accessed by various output commands. The per−atom values be accessed on any timestep.

No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions: none

Related commands:

dump custom, compute property/atom, variable

Default:

The option default is com = no.

fix temp/berendsen command

fix temp/berendsen/cuda command

Syntax:

```
fix ID group-ID temp/berendsen Tstart Tstop Tdamp
```

- ID, group−ID are documented in fix command
- temp/berendsen = style name of this fix command
- Tstart,Tstop = desired temperature at start/end of run
- Tdamp = temperature damping parameter (time units)

Examples:

```
fix 1 all temp/berendsen 300.0 300.0 100.0
```

Description:

Reset the temperature of a group of atoms by using a Berendsen thermostat (Berendsen), which rescales their velocities every timestep.

The thermostat is applied to only the translational degrees of freedom for the particles, which is an important consideration if extended spherical or aspherical particles which have rotational degrees of freedom are being thermostatted with this fix. The translational degrees of freedom can also have a bias velocity removed from them before thermostatting takes place; see the description below.

The desired temperature at each timestep is a ramped value during the run from *Tstart* to *Tstop*. The *Tdamp* parameter is specified in time units and determines how rapidly the temperature is relaxed. For example, a value of 100.0 means to relax the temperature in a timespan of (roughly) 100 time units (tau or fmsec or psec − see the units command).

IMPORTANT NOTE: Unlike the fix nvt command which performs Nose/Hoover thermostatting AND time integration, this fix does NOT perform time integration. It only modifies velocities to effect thermostatting. Thus you must use a separate time integration fix, like fix nve to actually update the positions of atoms using the modified velocities. Likewise, this fix should not normally be used on atoms that also have their temperature controlled by another fix − e.g. by fix nvt or fix langevin commands.

See this howto section of the manual for a discussion of different ways to compute temperature and perform thermostatting.

This fix computes a temperature each timestep. To do this, the fix creates its own compute of style "temp", as if this command had been issued:

```
compute fix-ID_temp group-ID temp
```

See the compute temp command for details. Note that the ID of the new compute is the fix−ID + underscore + "temp", and the group for the new compute is the same as the fix group.

Note that this is NOT the compute used by thermodynamic output (see the thermo_style command) with ID =

thermo_temp. This means you can change the attributes of this fix's temperature (e.g. its degrees−of−freedom) via the compute_modify command or print this temperature during thermodynamic output via the thermo_style custom command using the appropriate compute−ID. It also means that changing attributes of *thermo_temp* will have no effect on this fix.

Like other fixes that perform thermostatting, this fix can be used with compute commands that calculate a temperature after removing a "bias" from the atom velocities. E.g. removing the center−of−mass velocity from a group of atoms or only calculating temperature on the x−component of velocity or only calculating temperature for atoms in a geometric region. This is not done by default, but only if the fix_modify command is used to assign a temperature compute to this fix that includes such a bias term. See the doc pages for individual compute commands to determine which ones include a bias. In this case, the thermostat works in the following manner: the current temperature is calculated taking the bias into account, bias is removed from each atom, thermostatting is performed on the remaining thermal degrees of freedom, and the bias is added back in.

Styles with a *cuda* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in this section of the manual. The accelerated styles take the same arguments and should produce the same results, except for round−off and precision issues.

These accelerated styles are part of the "user−cuda" package. They are only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the −suffix command−line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See this section of the manual for more instructions on how to use the accelerated styles effectively.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files.

The fix_modify *temp* option is supported by this fix. You can use it to assign a temperature compute you have defined to this fix which will be used in its thermostatting procedure, as described above. For consistency, the group used by this fix and by the compute should be the same.

The fix_modify *energy* option is supported by this fix to add the energy change implied by a velocity rescaling to the system's potential energy as part of thermodynamic output.

This fix computes a global scalar which can be accessed by various output commands. The scalar is the cummulative energy change due to this fix. The scalar value calculated by this fix is "extensive".

This fix can ramp its target temperature over multiple runs, using the *start* and *stop* keywords of the run command. See the run command for details of how to do this.

This fix is not invoked during energy minimization.

Restrictions: none

Related commands:

fix nve, fix nvt, fix temp/rescale, fix langevin, fix_modify, compute temp, fix press/berendsen

Default: none

(Berendsen) Berendsen, Postma, van Gunsteren, DiNola, Haak, J Chem Phys, 81, 3684 (1984).

fix temp/rescale command

fix temp/rescale/cuda command

fix temp/rescale/limit/cuda command

Syntax:

```
fix ID group−ID temp/rescale N Tstart Tstop window fraction
```

- ID, group−ID are documented in fix command
- temp/rescale = style name of this fix command
- N = perform rescaling every N steps
- Tstart,Tstop = desired temperature at start/end of run (temperature units)
- window = only rescale if temperature is outside this window (temperature units)
- fraction = rescale to target temperature by this fraction

Examples:

```
fix 3 flow temp/rescale 100 1.0 1.1 0.02 0.5
fix 3 boundary temp/rescale 1 1.0 1.5 0.05 1.0
fix 3 boundary temp/rescale 1 1.0 1.5 0.05 1.0
```

Description:

Reset the temperature of a group of atoms by explicitly rescaling their velocities.

The rescaling is applied to only the translational degrees of freedom for the particles, which is an important consideration if extended spherical or aspherical particles which have rotational degrees of freedom are being thermostatted with this fix. The translational degrees of freedom can also have a bias velocity removed from them before thermostatting takes place; see the description below.

Rescaling is performed every N timesteps. The target temperature is a ramped value between the *Tstart* and *Tstop* temperatures at the beginning and end of the run.

Rescaling is only performed if the difference between the current and desired temperatures is greater than the *window* value. The amount of rescaling that is applied is a *fraction* (from 0.0 to 1.0) of the difference between the actual and desired temperature. E.g. if *fraction* = 1.0, the temperature is reset to exactly the desired value.

IMPORTANT NOTE: Unlike the fix nvt command which performs Nose/Hoover thermostatting AND time integration, this fix does NOT perform time integration. It only modifies velocities to effect thermostatting. Thus you must use a separate time integration fix, like fix nve to actually update the positions of atoms using the modified velocities. Likewise, this fix should not normally be used on atoms that also have their temperature controlled by another fix − e.g. by fix nvt or fix langevin commands.

See this howto section of the manual for a discussion of different ways to compute temperature and perform thermostatting.

This fix computes a temperature each timestep. To do this, the fix creates its own compute of style "temp", as if one of this command had been issued:

```
compute fix-ID_temp group-ID temp
```

See the compute temp for details. Note that the ID of the new compute is the fix−ID + underscore + "temp", and the group for the new compute is the same as the fix group.

Note that this is NOT the compute used by thermodynamic output (see the thermo_style command) with ID = *thermo_temp*. This means you can change the attributes of this fix's temperature (e.g. its degrees−of−freedom) via the compute_modify command or print this temperature during thermodynamic output via the thermo_style custom command using the appropriate compute−ID. It also means that changing attributes of *thermo_temp* will have no effect on this fix.

Like other fixes that perform thermostatting, this fix can be used with compute commands that calculate a temperature after removing a "bias" from the atom velocities. E.g. removing the center−of−mass velocity from a group of atoms or only calculating temperature on the x−component of velocity or only calculating temperature for atoms in a geometric region. This is not done by default, but only if the fix_modify command is used to assign a temperature compute to this fix that includes such a bias term. See the doc pages for individual compute commands to determine which ones include a bias. In this case, the thermostat works in the following manner: the current temperature is calculated taking the bias into account, bias is removed from each atom, thermostatting is performed on the remaining thermal degrees of freedom, and the bias is added back in.

Styles with a *cuda* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in this section of the manual. The accelerated styles take the same arguments and should produce the same results, except for round−off and precision issues.

These accelerated styles are part of the "user−cuda" package. They are only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the −suffix command−line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See this section of the manual for more instructions on how to use the accelerated styles effectively.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files.

The fix_modify *temp* option is supported by this fix. You can use it to assign a temperature compute you have defined to this fix which will be used in its thermostatting procedure, as described above. For consistency, the group used by this fix and by the compute should be the same.

The fix_modify *energy* option is supported by this fix to add the energy change implied by a velocity rescaling to the system's potential energy as part of thermodynamic output.

This fix computes a global scalar which can be accessed by various output commands. The scalar is the cummulative energy change due to this fix. The scalar value calculated by this fix is "extensive".

This fix can ramp its target temperature over multiple runs, using the *start* and *stop* keywords of the run command. See the run command for details of how to do this.

This fix is not invoked during energy minimization.

Restrictions: none

Related commands:

fix langevin, fix nvt, fix_modify

Default: none

fix temp/rescale/eff command

Syntax:

```
fix ID group−ID temp/rescale/eff N Tstart Tstop window fraction
```

- ID, group−ID are documented in fix command
- temp/rescale/eff = style name of this fix command
- N = perform rescaling every N steps
- Tstart,Tstop = desired temperature at start/end of run (temperature units)
- window = only rescale if temperature is outside this window (temperature units)
- fraction = rescale to target temperature by this fraction

Examples:

```
fix 3 flow temp/rescale/eff 10 1.0 100.0 0.02 1.0
```

Description:

Reset the temperature of a group of nuclei and electrons in the electron force field model by explicitly rescaling their velocities.

The operation of this fix is exactly like that described by the fix temp/rescale command, except that the rescaling is also applied to the radial electron velocity for electron particles.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files.

The fix_modify *temp* option is supported by this fix. You can use it to assign a temperature compute you have defined to this fix which will be used in its thermostatting procedure, as described above. For consistency, the group used by this fix and by the compute should be the same.

The fix_modify *energy* option is supported by this fix to add the energy change implied by a velocity rescaling to the system's potential energy as part of thermodynamic output.

This fix computes a global scalar which can be accessed by various output commands. The scalar is the cummulative energy change due to this fix. The scalar value calculated by this fix is "extensive".

This fix can ramp its target temperature over multiple runs, using the *start* and *stop* keywords of the run command. See the run command for details of how to do this.

This fix is not invoked during energy minimization.

Restrictions:

This fix is part of the "user−eff" package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

Related commands:

fix langevin/eff, fix nvt/eff, fix_modify, fix_temp_rescale,

Default: none

fix thermal/conductivity command

Syntax:

```
fix ID group-ID thermal/conductivity N edim Nbin keyword value ...
```

- ID, group−ID are documented in fix command
- thermal/conductivity = style name of this fix command
- N = perform kinetic energy exchange every N steps
- edim = *x* or *y* or *z* = direction of kinetic energy transfer
- Nbin = # of layers in edim direction (must be even number)
- zero or more keyword/value pairs may be appended
- keyword = *swap*

```
  swap value = Nswap = number of swaps to perform every N steps
```

Examples:

```
fix 1 all thermal/conductivity 100 z 20
fix 1 all thermal/conductivity 50 z 20 swap 2
```

Description:

Use the Muller−Plathe algorithm described in this paper to exchange kinetic energy between two particles in different regions of the simulation box every N steps. This induces a temperature gradient in the system. As described below this enables a thermal conductivity of the fluid to be calculated. This algorithm is sometimes called a reverse non−equilibrium MD (reverse NEMD) approach to computing thermal conductivity. This is because the usual NEMD approach is to impose a temperature gradient on the system and measure the response as the resulting heat flux. In the Muller−Plathe method, the heat flux is imposed, and the temperature gradient is the system's response.

See the compute heat/flux command for details on how to compute thermal conductivity in an alternate way, via the Green−Kubo formalism.

The simulation box is divided into *Nbin* layers in the *edim* direction, where the layer 1 is at the low end of that dimension and the layer *Nbin* is at the high end. Every N steps, Nswap pairs of atoms are chosen in the following manner. Only atoms in the fix group are considered. The hottest Nswap atoms in layer 1 are selected. Similarly, the coldest Nswap atoms in the "middle" layer (see below) are selected. The two sets of Nswap atoms are paired up and their velocities are exchanged. This effectively swaps their kinetic energies, assuming their masses are the same. Over time, this induces a temperature gradient in the system which can be measured using commands such as the following, which writes the temperature profile (assuming z = edim) to the file tmp.profile:

```
compute   ke all ke/atom
variable  temp atom c_ke/1.5
fix       3 all ave/spatial 10 100 1000 z lower 0.05 v_temp &           file tmp.profile units redu
```

Note that by default, Nswap = 1, though this can be changed by the optional *swap* keyword. Setting this parameter appropriately, in conjunction with the swap rate N, allows the heat flux to be adjusted across a wide range of values, and the kinetic energy to be exchanged in large chunks or more smoothly.

The "middle" layer for velocity swapping is defined as the *Nbin*/2 + 1 layer. Thus if *Nbin* = 20, the two swapping

layers are 1 and 11. This should lead to a symmetric temperature profile since the two layers are separated by the same distance in both directions in a periodic sense. This is why *Nbin* is restricted to being an even number.

As described below, the total kinetic energy transferred by these swaps is computed by the fix and can be output. Dividing this quantity by time and the cross−sectional area of the simulation box yields a heat flux. The ratio of heat flux to the slope of the temperature profile is the thermal conductivity of the fluid, in appopriate units. See the Muller−Plathe paper for details.

IMPORTANT NOTE: After equilibration, if the temperature gradient you observe is not linear, then you are likely swapping energy too frequently and are not in a regime of linear response. In this case you cannot accurately infer a thermal conductivity and should try increasing the Nevery parameter.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix.

This fix computes a global scalar which can be accessed by various output commands. The scalar is the cummulative kinetic energy transferred between the bottom and middle of the simulation box (in the *edim* direction) is stored as a scalar quantity by this fix. This quantity is zeroed when the fix is defined and accumlates thereafter, once every N steps. The units of the quantity are energy; see the units command for details. The scalar value calculated by this fix is "intensive".

No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions:

Swaps conserve both momentum and kinetic energy, even if the masses of the swapped atoms are not equal. Thus you should not need to thermostat the system. If you do use a thermostat, you may want to apply it only to the non−swapped dimensions (other than *vdim*).

LAMMPS does not check, but you should not use this fix to swap the kinetic energy of atoms that are in constrained molecules, e.g. via fix shake or fix rigid. This is because application of the constraints will alter the amount of transferred momentum. You should, however, be able to use flexible molecules. See the Zhang paper for a discussion and results of this idea.

When running a simulation with large, massive particles or molecules in a background solvent, you may want to only exchange kinetic energy bewteen solvent particles.

Related commands:

fix ave/spatial, fix viscosity, compute heat/flux

Default:

The option defaults are swap = 1.

(Muller−Plathe) Muller−Plathe, J Chem Phys, 106, 6082 (1997).

(Zhang) Zhang, Lussetti, de Souza, Muller−Plathe, J Phys Chem B, 109, 15060−15067 (2005).

fix tmd command

Syntax:

```
fix ID group-ID tmd rho_final file1 N file2
```

- ID, group−ID are documented in fix command
- tmd = style name of this fix command
- rho_final = desired value of rho at the end of the run (distance units)
- file1 = filename to read target structure from
- N = dump TMD statistics every this many timesteps, 0 = no dump
- file2 = filename to write TMD statistics to (only needed if N > 0)

Examples:

```
fix 1 all nve
fix 2 tmdatoms tmd 1.0 target_file 100 tmd_dump_file
```

Description:

Perform targeted molecular dynamics (TMD) on a group of atoms. A holonomic constraint is used to force the atoms to move towards (or away from) the target configuration. The parameter "rho" is monotonically decreased (or increased) from its initial value to rho_final at the end of the run.

Rho has distance units and is a measure of the root−mean−squared distance (RMSD) between the current configuration of the atoms in the group and the target coordinates listed in file1. Thus a value of rho_final = 0.0 means move the atoms all the way to the final structure during the course of the run.

The target file1 can be ASCII text or a gzipped text file (detected by a .gz suffix). The format of the target file1 is as follows:

```
0.0 25.0 xlo xhi
0.0 25.0 ylo yhi
0.0 25.0 zlo zhi
125     24.97311   1.69005     23.46956 0 0 -1
126     1.94691    2.79640     1.92799  1 0 0
127     0.15906    3.46099     0.79121  1 0 0
...
```

The first 3 lines may or may not be needed, depending on the format of the atoms to follow. If image flags are included with the atoms, the 1st 3 lo/hi lines must appear in the file. If image flags are not included, the 1st 3 lines should not appear. The 3 lines contain the simulation box dimensions for the atom coordinates, in the same format as in a LAMMPS data file (see the read_data command).

The remaining lines each contain an atom ID and its target x,y,z coordinates. The atom lines (all or none of them) can optionally be followed by 3 integer values: nx,ny,nz. For periodic dimensions, they specify which image of the box the atom is considered to be in, i.e. a value of N (positive or negative) means add N times the box length to the coordinate to get the true value.

The atom lines can be listed in any order, but every atom in the group must be listed in the file. Atoms not in the fix group may also be listed; they will be ignored.

TMD statistics are written to file2 every N timesteps, unless N is specified as 0, which means no statistics.

The atoms in the fix tmd group should be integrated (via a fix nve, nvt, npt) along with other atoms in the system.

Restarts can be used with a fix tmd command. For example, imagine a 10000 timestep run with a rho_initial = 11 and a rho_final = 1. If a restart file was written after 2000 time steps, then the configuration in the file would have a rho value of 9. A new 8000 time step run could be performed with the same rho_final = 1 to complete the conformational change at the same transition rate. Note that for restarted runs, the name of the TMD statistics file should be changed to prevent it being overwritten.

For more information about TMD, see (Schlitter1) and (Schlitter2).

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix. No global or per−atom quantities are stored by this fix for access by various output commands.

This fix can ramp its rho parameter over multiple runs, using the *start* and *stop* keywords of the run command. See the run command for details of how to do this.

This fix is not invoked during energy minimization.

Restrictions:

All TMD fixes must be listed in the input script after all integrator fixes (nve, nvt, npt) are applied. This ensures that atoms are moved before their positions are corrected to comply with the constraint.

Atoms that have a TMD fix applied should not be part of a group to which a SHAKE fix is applied. This is because LAMMPS assumes there are not multiple competing holonomic constraints applied to the same atoms.

To read gzipped target files, you must compile LAMMPS with the −DLAMMPS_GZIP option − see the Making LAMMPS section of the documentation.

Related commands: none

Default: none

(Schlitter1) Schlitter, Swegat, Mulders, "Distance−type reaction coordinates for modelling activated processes", J Molecular Modeling, 7, 171−177 (2001).

(Schlitter2) Schlitter and Klahn, "The free energy of a reaction coordinate at multiple constraints: a concise formulation", Molecular Physics, 101, 3439−3443 (2003).

fix ttm command

Syntax:

```
fix ID group-ID ttm seed C_e rho_e kappa_e gamma_p gamma_s v_0 Nx Ny Nz T_infile N T_outfile
```

- ID, group−ID are documented in fix command
- ttm = style name of this fix command
- seed = random number seed to use for white noise (positive integer)
- C_e = electronic specific heat (energy/(electron*temperature) units)
- rho_e = electronic density (electrons/volume units)
- kappa_e = electronic thermal conductivity (energy/(time*distance*temperature) units)
- gamma_p = friction coefficient due to electron−ion interactions (mass/time units)
- gamma_s = friction coefficient due to electronic stopping (mass/time units)
- v_0 = electronic stopping critical velocity (velocity units)
- Nx = number of thermal solve grid points in the x−direction (positive integer)
- Ny = number of thermal solve grid points in the y−direction (positive integer)
- Nz = number of thermal solve grid points in the z−direction (positive integer)
- T_infile = filename to read initial electronic temperature from
- N = dump TTM temperatures every this many timesteps, 0 = no dump
- T_outfile = filename to write TTM temperatures to (only needed if N > 0)

Examples:

```
fix 2 all ttm 699489 1.0 1.0 10 0.1 0.0 2.0 1 12 1 initialTs 1000 T.out
fix 2 all ttm 123456 1.0 1.0 1.0 1.0 1.0 5.0 5 5 5 Te.in 1 Te.out
```

Description:

Use a two−temperature model (TTM) to represent heat transfer through and between electronic and atomic subsystems. LAMMPS models the atomic subsystem as usual with a molecular dynamics model and the classical force field specified by the user, but the electronic subsystem is modeled as a continuum, or a background "gas", on a regular grid. Energy can be transferred spatially within the grid representing the electrons. Energy can also be transferred between the electronic and the atomic subsystems. The algorithm underlying this fix was derived by D. M. Duffy and A. M. Rutherford and is discussed in two J Physics: Condensed Matter papers: (Duffy) and (Rutherford). They used this algorithm in cascade simulations where a primary knock−on atom (PKA) was initialized with a high velocity to simulate a radiation event.

Heat transfer between the electronic and atomic subsystems is carried out via an inhomogeneous Langevin thermostat. This thermostat differs from the regular Langevin thermostat (fix langevin) in three important ways. First, the Langevin thermostat is applied uniformly to all atoms in the user−specified group for a single target temperature, whereas the TTM fix applies Langevin thermostatting locally to atoms within the volumes represented by the user−specified grid points with a target temperature specific to that grid point. Second, the Langevin thermostat couples the temperature of the atoms to an infinite heat reservoir, whereas the heat reservoir for fix TTM is finite and represents the local electrons. Third, the TTM fix allows users to specify not just one friction coefficient, but rather two independent friction coefficients: one for the electron−ion interactions (*gamma_p*), and one for electron stopping (*gamma_s*).

When the friction coefficient due to electron stopping, *gamma_s*, is non−zero, electron stopping effects are included for atoms moving faster than the electron stopping critical velocity, *v_0*. For further details about this

algorithm, see (Duffy) and (Rutherford).

Energy transport within the electronic subsystem is solved according to the heat diffusion equation with added source terms for heat transfer between the subsystems:

$$C_e \rho_e \frac{\partial T_e}{\partial t} = \bigtriangledown (\kappa_e \bigtriangledown T_e) - g_p (T_e - T_a) + g_s T'_a$$

where C_e is the specific heat, rho_e is the density, kappa_e is the thermal conductivity, T is temperature, the "e" and "a" subscripts represent electronic and atomic subsystems respectively, g_p is the coupling constant for the electron−ion interaction, and g_s is the electron stopping coupling parameter. C_e, rho_e, and kappa_e are specified as parameters to the fix. The other quantities are derived. The form of the heat diffusion equation used here is almost the same as that in equation 6 of (Duffy), with the exception that the electronic density is explicitly reprensented, rather than being part of the the specific heat parameter.

Currently, this fix assumes that none of the user−supplied parameters will vary with temperature. This assumption can be relaxed by modifying the source code to include the desired temperature dependency and functional form for any of the parameters. Note that (Duffy) used a tanh() functional form for the temperature dependence of the electronic specific heat, but ignored temperature dependencies of any of the other parameters.

This fix requires use of periodic boundary conditions and a 3D simulation. Periodic boundary conditions are also used in the heat equation solve for the electronic subsystem. This varies from the approach of (Rutherford) where the atomic subsystem was embedded within a larger continuum representation of the electronic subsystem.

The initial electronic temperature input file, *T_infile*, is a text file LAMMPS reads in with no header and with four numeric columns (ix,iy,iz,Temp) and with a number of rows equal to the number of user−specified grid points (Nx by Ny by Nz). The ix,iy,iz are node indices from 0 to nxnodes−1, etc. For example, the initial electronic temperatures on a 1 by 2 by 3 grid could be specified in a *T_infile* as follows:

```
0 0 0 1.0
0 0 1 1.0
0 0 2 1.0
0 1 0 2.0
0 1 1 2.0
0 1 2 2.0
```

where the electronic temperatures along the y=0 plane have been set to 1.0, and the electronic temperatures along the y=1 plane have been set to 2.0. The order of lines in this file is no important. If all the nodal values are not specified, LAMMPS will generate an error.

The temperature output file, *T_oufile*, is created and written by this fix. Temperatures for both the electronic and atomic subsystems at every node and every N timesteps are output. If N is specified as zero, no output is generated, and no output filename is needed. The format of the output is as follows. One long line is written every output timestep. The timestep itself is given in the first column. The next Nx*Ny*Nz columns contain the temperatures for the atomic subsystem, and the final Nx*Ny*Nz columns contain the temperatures for the electronic subsystem. The ordering of the Nx*Ny*Nz columns is with the z index varing fastest, y the next fastest, and x the slowest.

This fix does not change the coordinates of its atoms; it only scales their velocities. Thus a time integration fix (e.g. fix nve) should still be used to time integrate the affected atoms. This fix should not normally be used on atoms that have their temperature controlled by another fix − e.g. fix nvt or fix langevin.

This fix computes 2 output quantities stored in a vector of length 2, which can be accessed by various output commands. The first quantity is the total energy of the electronic subsystem. The second quantity is the energy transferred from the electronic to the atomic subsystem on that timestep. Note that the velocity verlet integrator applies the fix ttm forces to the atomic subsystem as two half−step velocity updates: one on the current timestep and one on the subsequent timestep. Consequently, the change in the atomic subsystem energy is lagged by half a timestep relative to the change in the electronic subsystem energy. As a result of this, users may notice slight fluctuations in the sum of the atomic and electronic subsystem energies reported at the end of the timestep.

The vector values calculated by this fix are "extensive".

IMPORTANT NOTE: The current implementation creates a copy of the electron grid that overlays the entire simulation domain, for each processor. Values on the grid are summed across all processors. Thus you should insure that this grid is not too large, else your simulation could incur high memory and communication costs.

Restart, fix_modify, output, run start/stop, minimize info:

This fix writes the state of the electronic subsystem and the energy exchange between the subsystems to binary restart files. See the read_restart command for info on how to re−specify a fix in an input script that reads a restart file, so that the operation of the fix continues in an uninterrupted fashion.

Because the state of the random number generator is not saved in the restart files, this means you cannot do "exact" restarts with this fix, where the simulation continues on the same as if no restart had taken place. However, in a statistical sense, a restarted simulation should produce the same behavior.

None of the fix_modify options are relevant to this fix. No global or per−atom quantities are stored by this fix for access by various output commands. No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions:

This fix can only be used for 3d simulations and orthogonal simlulation boxes. You must use periodic boundary conditions with this fix.

Related commands:

fix langevin, fix dt/reset

Default: none

(Duffy) D M Duffy and A M Rutherford, J. Phys.: Condens. Matter, 19, 016207−016218 (2007).

(Rutherford) A M Rutherford and D M Duffy, J. Phys.: Condens. Matter, 19, 496201−496210 (2007).

fix viscosity command

Syntax:

```
fix ID group-ID viscosity N vdim pdim Nbin keyword value ...
```

- ID, group−ID are documented in fix command
- viscosity = style name of this fix command
- N = perform momentum exchange every N steps
- vdim = *x* or *y* or *z* = which momentum component to exchange
- pdim = *x* or *y* or *z* = direction of momentum transfer
- Nbin = # of layers in pdim direction (must be even number)
- zero or more keyword/value pairs may be appended
- keyword = *swap* or *target*

```
  swap value = Nswap = number of swaps to perform every N steps
    vtarget value = V or INF = target velocity of swap partners (velocity units)
```

Examples:

```
fix 1 all viscosity 100 x z 20
fix 1 all viscosity 50 x z 20 swap 2 vtarget 1.5
```

Description:

Use the Muller−Plathe algorithm described in this paper to exchange momenta between two particles in different regions of the simulation box every N steps. This induces a shear velocity profile in the system. As described below this enables a viscosity of the fluid to be calculated. This algorithm is sometimes called a reverse non−equilibrium MD (reverse NEMD) approach to computing viscosity. This is because the usual NEMD approach is to impose a shear velocity profile on the system and measure the response via an off−diagonal component of the stress tensor, which is proportional to the momentum flux. In the Muller−Plathe method, the momentum flux is imposed, and the shear velocity profile is the system's response.

The simulation box is divided into *Nbin* layers in the *pdim* direction, where the layer 1 is at the low end of that dimension and the layer *Nbin* is at the high end. Every N steps, Nswap pairs of atoms are chosen in the following manner. Only atoms in the fix group are considered. Nswap atoms in layer 1 with positive velocity components in the *vdim* direction closest to the target value *V* are selected. Similarly, Nswap atoms in the "middle" layer (see below) with negative velocity components in the *vdim* direction closest to the negative of the target value *V* are selected. The two sets of Nswap atoms are paired up and their *vdim* momenta components are swapped within each pair. This resets their velocities, typically in opposite directions. Over time, this induces a shear velocity profile in the system which can be measured using commands such as the following, which writes the profile to the file tmp.profile:

```
fix f1 all ave/spatial 100 10 1000 z lower 0.05 vx &    file tmp.profile units reduced
```

Note that by default, Nswap = 1 and vtarget = INF, though this can be changed by the optional *swap* and *vtarget* keywords. When vtarget = INF, one or more atoms with the most positive and negative velocity components are selected. Setting these parameters appropriately, in conjunction with the swap rate N, allows the momentum flux rate to be adjusted across a wide range of values, and the momenta to be exchanged in large chunks or more smoothly.

The "middle" layer for momenta swapping is defined as the *Nbin*/2 + 1 layer. Thus if *Nbin* = 20, the two swapping layers are 1 and 11. This should lead to a symmetric velocity profile since the two layers are separated by the same distance in both directions in a periodic sense. This is why *Nbin* is restricted to being an even number.

As described below, the total momentum transferred by these velocity swaps is computed by the fix and can be output. Dividing this quantity by time and the cross−sectional area of the simulation box yields a momentum flux. The ratio of momentum flux to the slope of the shear velocity profile is the viscosity of the fluid, in appopriate units. See the Muller−Plathe paper for details.

IMPORTANT NOTE: After equilibration, if the velocity profile you observe is not linear, then you are likely swapping momentum too frequently and are not in a regime of linear response. In this case you cannot accurately infer a viscosity and should try increasing the Nevery parameter.

An alternative method for calculating a viscosity is to run a NEMD simulation, as described in this section of the manual. NEMD simulations deform the simmulation box via the fix deform command. Thus they cannot be run on a charged system using a PPPM solver since PPPM does not currently support non−orthogonal boxes. Using fix viscosity keeps the box orthogonal; thus it does not suffer from this limitation.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix.

This fix computes a global scalar which can be accessed by various output commands. The scalar is the cummulative momentum transferred between the bottom and middle of the simulation box (in the *pdim* direction) is stored as a scalar quantity by this fix. This quantity is zeroed when the fix is defined and accumlates thereafter, once every N steps. The units of the quantity are momentum = mass*velocity. The scalar value calculated by this fix is "intensive".

No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions:

Swaps conserve both momentum and kinetic energy, even if the masses of the swapped atoms are not equal. Thus you should not need to thermostat the system. If you do use a thermostat, you may want to apply it only to the non−swapped dimensions (other than *vdim*).

LAMMPS does not check, but you should not use this fix to swap velocities of atoms that are in constrained molecules, e.g. via fix shake or fix rigid. This is because application of the constraints will alter the amount of transferred momentum. You should, however, be able to use flexible molecules. See the Maginn paper for an example of using this algorithm in a computation of alcohol molecule properties.

When running a simulation with large, massive particles or molecules in a background solvent, you may want to only exchange momenta bewteen solvent particles.

Related commands:

fix ave/spatial, fix thermal/conductivity

Default:

The option defaults are swap = 1 and vtarget = INF.

(Muller−Plathe) Muller−Plathe, Phys Rev E, 59, 4894−4898 (1999).

(Maginn) Kelkar, Rafferty, Maginn, Siepmann, Fluid Phase Equilibria, 260, 218−231 (2007).

fix viscous command

fix viscous/cuda command

Syntax:

```
fix ID group-ID viscous gamma keyword values ...
```

- ID, group−ID are documented in fix command
- viscous = style name of this fix command
- gamma = damping coefficient (force/velocity units)
- zero or more keyword/value pairs may be appended

```
keyword = scale
  scale values = type ratio
    type = atom type (1-N)
    ratio = factor to scale the damping coefficient by
```

Examples:

```
fix 1 flow viscous 0.1
fix 1 damp viscous 0.5 scale 3 2.5
```

Description:

Add a viscous damping force to atoms in the group that is proportional to the velocity of the atom. The added force can be thought of as a frictional interaction with implicit solvent, i.e. the no−slip Stokes drag on a spherical particle. In granular simulations this can be useful for draining the kinetic energy from the system in a controlled fashion. If used without additional thermostatting (to add kinetic energy to the system), it has the effect of slowly (or rapidly) freezing the system; hence it can also be used as a simple energy minimization technique.

The damping force F is given by F = − gamma * velocity. The larger the coefficient, the faster the kinetic energy is reduced. If the optional keyword *scale* is used, gamma can scaled up or down by the specified factor for atoms of that type. It can be used multiple times to adjust gamma for several atom types.

IMPORTANT NOTE: You should specify gamma in force/velocity units. This is not the same as mass/time units, at least for some of the LAMMPS units options like "real" or "metal" that are not self−consistent.

In a Brownian dynamics context, gamma = Kb T / D, where Kb = Boltzmann's constant, T = temperature, and D = particle diffusion coefficient. D can be written as Kb T / (3 pi eta d), where eta = dynamic viscosity of the frictional fluid and d = diameter of particle. This means gamma = 3 pi eta d, and thus is proportional to the viscosity of the fluid and the particle diameter.

In the current implementation, rather than have the user specify a viscosity, gamma is specified directly in force/velocity units. If needed, gamma can be adjusted for atoms of different sizes (i.e. sigma) by using the *scale* keyword.

Note that Brownian dynamics models also typically include a randomized force term to thermostat the system at a chosen temperature. The fix langevin command does this. It has the same viscous damping term as fix viscous and adds a random force to each atom. Hence if using fix *langevin* you do not typically need to use fix *viscous*. Also note that the gamma of fix viscous is related to the damping parameter of fix langevin, except that the units

of gamma are force/velocity and the units of damp are time, so that it can more easily be used as a thermostat.

Styles with a *cuda* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in this section of the manual. The accelerated styles take the same arguments and should produce the same results, except for round−off and precision issues.

These accelerated styles are part of the "user−cuda" package. They are only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the −suffix command−line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See this section of the manual for more instructions on how to use the accelerated styles effectively.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix. No global or per−atom quantities are stored by this fix for access by various output commands. No parameter of this fix can be used with the *start/stop* keywords of the run command.

The forces due to this fix are imposed during an energy minimization, invoked by the minimize command. This fix should only be used with damped dynamics minimizers that allow for non−conservative forces. See the min_style command for details.

Restrictions: none

Related commands:

fix langevin

Default: none

fix wall/lj93 command

fix wall/lj126 command

fix wall/colloid command

fix wall/harmonic command

Syntax:

```
fix ID group-ID style face args ... keyword value ...
```

- ID, group−ID are documented in fix command
- style = *wall/lj93* or *wall/lj126* or *wall/colloid* or *wall/harmonic*
- one or more face/arg pairs may be appended
- face = *xlo* or *xhi* or *ylo* or *yhi* or *zlo* or *zhi*

```
  args = coord epsilon sigma cutoff
    coord = position of wall = EDGE or constant or variable
      EDGE = current lo or hi edge of simulation box
      constant = number like 0.0 or -30.0 (distance units)
      variable = equal-style variable like v_x or v_wiggle
    epsilon = strength factor for wall-particle interaction (energy or energy/distance^2 units
    sigma = size factor for wall-particle interaction (distance units)
    cutoff = distance from wall at which wall-particle interaction is cut off (distance units)
```

- zero or more keyword/value pairs may be appended
- keyword = *units*

```
units value = lattice or box
    lattice = the wall position is defined in lattice units
    box = the wall position is defined in simulation box units
```

Examples:

```
fix wallhi all wall/lj93 xlo -1.0 1.0 1.0 2.5 units box
fix wallhi all wall/lj93 xhi EDGE 1.0 1.0 2.5
fix wallhi all wall/lj126 v_wiggle 23.2 1.0 1.0 2.5
fix zwalls all wall/colloid zlo 0.0 1.0 1.0 0.858 zhi 40.0 1.0 1.0 0.858
```

Description:

Bound the simulation domain on one or more of its faces with a flat wall that interacts with the atoms in the group by generating a force on the atom in a direction perpendicular to the wall. The energy of wall−particle interactions depends on the style.

For style *wall/lj93*, the energy E is given by the 9/3 potential:

$$E = \epsilon \left[\frac{2}{15} \left(\frac{\sigma}{r} \right)^9 - \left(\frac{\sigma}{r} \right)^3 \right] \qquad r < r_c$$

For style *wall/lj126*, the energy E is given by the 12/6 potential:

$$E = 4\epsilon \left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right] \qquad r < r_c$$

For style *wall/colloid*, the energy E is given by an integrated form of the pair_style colloid potential:

$$E = 144\phi^2\epsilon \left[\frac{\sigma^6}{7560} \left(\frac{6R-D}{D^7} + \frac{D+8R}{(D+2R)^7} \right) - \frac{1}{6} \left(\frac{2R(D+R) + D(D+2R)\left[\ln D - \ln(D+2R)\right]}{D(D+2R)} \right) \right] \qquad r < r_c$$

For style *wall/harmonic*, the energy E is given by a harmonic spring potential:

$$E = \epsilon \left(r - r_c\right)^2 \qquad r < r_c$$

In all cases, *r* is the distance from the particle to the wall at position *coord*, and Rc is the *cutoff* distance at which the particle and wall no longer interact. The energy of the wall potential is shifted so that the wall−particle interaction energy is 0.0 at the cutoff distance.

Up to 6 walls or faces can be specified in a single command: *xlo*, *xhi*, *ylo*, *yhi*, *zlo*, *zhi*. A *lo* face interacts with particles near the lower side of the simulation box in that dimension. A *hi* face interacts with particles near the upper side of the simulation box in that dimension.

The position of each wall can be specified in one of 3 ways: as the EDGE of the simulation box, as a constant value, or as a variable. If EDGE is used, then the corresponding boundary of the current simulation box is used. If a numeric constant is specified then the wall is placed at that position in the appropriate dimension (x, y, or z). In both the EDGE and constant cases, the wall will never move. If the wall position is a variable, it should be specified as v_name, where name is an equal−style variable name. In this case the variable is evaluated each timestep and the result becomes the current position of the reflecting wall. Equal−style variables can specify formulas with various mathematical functions, and include thermo_style command keywords for the simulation box parameters and timestep and elapsed time. Thus it is easy to specify a time−dependent wall position. See examples below.

For the *wall/lj93* and *wall/lj126* styles, *epsilon* and *sigma* are the usual Lennard−Jones parameters, which determine the strength and size of the particle as it interacts with the wall. Epsilon has energy units. Note that this *epsilon* and *sigma* may be different than any *epsilon* or *sigma* values defined for a pair style that computes particle−particle interactions.

The *wall/lj93* interaction is derived by integrating over a 3d half−lattice of Lennard−Jones 12/6 particles. The *wall/lj126* interaction is effectively a harder, more repulsive wall interaction.

For the *wall/colloid* style, *epsilon* is effectively a Hamaker constant with energy units for the colloid−wall interaction, *R* is the radius of the colloid particle, *D* is the distance from the surface of the colloid particle to the wall (r−R), and *sigma* is the size of a constituent LJ particle inside the colloid particle. Note that the cutoff distance Rc in this case is the distance from the colloid particle center to the wall.

The *wall/colloid* interaction is derived by integrating over constituent LJ particles of size *sigma* within the colloid particle and a 3d half−lattice of Lennard−Jones 12/6 particles of size *sigma* in the wall.

For the *wall/harmonic* style, *epsilon* is effectively the spring constant K, and has units (energy/distance^2). The input parameter *sigma* is ignored. The minimum energy position of the harmonic spring is at the *cutoff*. This is a repulsive−only spring since the interaction is truncated at the *cutoff*

IMPORTANT NOTE: For all of the styles, you must insure that r is always > 0 for all particles in the group, or LAMMPS will generate an error. This means you cannot start your simulation with particles at the wall position *coord* (r = 0) or with particles on the wrong side of the wall (r < 0). For the *wall/lj93* and *wall/lj126* styles, the energy of the wall/particle interaction (and hence the force on the particle) blows up as r −> 0. The *wall/colloid* style is even more restrictive, since the energy blows up as D = r−R −> 0. This means the finite−size particles of radius R must be a distance larger than R from the wall position *coord*. The *harmonic* style is a softer potential and does not blow up as r −> 0, but you must use a large enough *epsilon* that particles always reamin on the correct side of the wall (r > 0).

The *units* keyword determines the meaning of the distance units used to define a wall position, but only when a numeric constant is used. It is not relevant when EDGE or a variable is used to specify a face position.

A *box* value selects standard distance units as defined by the units command, e.g. Angstroms for units = real or metal. A *lattice* value means the distance units are in lattice spacings. The lattice command must have been previously used to define the lattice spacings.

Here are examples of variable definitions that move the wall position in a time−dependent fashion using equal−style variables.

```
variable ramp equal ramp(0,10)
fix 1 all wall xlo v_ramp 1.0 1.0 2.5

variable linear equal vlinear(0,20)
fix 1 all wall xlo v_linear 1.0 1.0 2.5

variable wiggle equal swiggle(0.0,5.0,3.0)
fix 1 all wall xlo v_wiggle 1.0 1.0 2.5

variable wiggle equal cwiggle(0.0,5.0,3.0)
fix 1 all wall xlo v_wiggle 1.0 1.0 2.5
```

The ramp(lo,hi) function adjusts the wall position linearly from lo to hi over the course of a run. The linear(c0,velocity) function does something similar using the equation position = c0 + velocity*delta, where delta is the elapsed time.

The swiggle(c0,A,period) function causes the wall position to oscillate sinusoidally according to this equation, where omega = 2 PI / period:

```
position = c0 + A sin(omega*delta)
```

The cwiggle(c0,A,period) function causes the wall position to oscillate sinusoidally according to this equation, which will have an initial wall velocity of 0.0, and thus may impose a gentler perturbation on the particles:

```
position = c0 + A (1 − cos(omega*delta))
```

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files.

The fix_modify *energy* option is supported by this fix to add the energy of interaction between atoms and each wall to the system's potential energy as part of thermodynamic output.

This fix computes a global scalar energy and a global vector of forces, which can be accessed by various output commands. Note that the scalar energy is the sum of interactions with all defined walls. If you want the energy on a per−wall basis, you need to use multiple fix wall commands. The length of the vector is equal to the number of walls defined by the fix. Each vector value is the normal force on a specific wall. Note that an outward force on a wall will be a negative value for *lo* walls and a positive value for *hi* walls. The scalar and vector values calculated by this fix are "extensive".

No parameter of this fix can be used with the *start/stop* keywords of the run command.

The forces due to this fix are imposed during an energy minimization, invoked by the minimize command.

IMPORTANT NOTE: If you want the atom/wall interaction energy to be included in the total potential energy of the system (the quantity being minimized), you MUST enable the fix_modify *energy* option for this fix.

Restrictions:

Any dimension (xyz) that has a wall must be non−periodic.

Related commands:

fix wall/reflect, fix wall/gran, fix wall/region

Default:

The option defaults are no velocity, no wiggle, and units = lattice.

fix wall/gran command

Syntax:

```
fix ID group−ID wall/gran Kn Kt gamma_n gamma_t xmu dampflag wallstyle args keyword values ...
```

- ID, group−ID are documented in fix command
- wall/gran = style name of this fix command
- Kn = elastic constant for normal particle repulsion (force/distance units or pressure units − see discussion below)
- Kt = elastic constant for tangential contact (force/distance units or pressure units − see discussion below)
- gamma_n = damping coefficient for collisions in normal direction (1/time units or 1/time−distance units − see discussion below)
- gamma_t = damping coefficient for collisions in tangential direction (1/time units or 1/time−distance units − see discussion below)
- xmu = static yield criterion (unitless fraction between 0.0 and 1.0)
- dampflag = 0 or 1 if tangential damping force is excluded or included
- wallstyle = *xplane* or *yplane* or *zplane* or *zcylinder*
- args = list of arguments for a particular style

```
  xplane or yplane or zplane args = lo hi
      lo,hi = position of lower and upper plane (distance units), either can be NULL)
    zcylinder args = radius
      radius = cylinder radius (distance units)
```

- zero or more keyword/value pairs may be appended to args
- keyword = *wiggle* or *shear*

```
  wiggle values = dim amplitude period
      dim = x or y or z
      amplitude = size of oscillation (distance units)
      period = time of oscillation (time units)
    shear values = dim vshear
      dim = x or y or z
      vshear = magnitude of shear velocity (velocity units)
```

Examples:

```
fix 1 all wall/gran 200000.0 NULL 50.0 NULL 0.5 0 xplane −10.0 10.0
fix 1 all wall/gran 200000.0 NULL 50.0 NULL 0.5 0 zplane 0.0 NULL
fix 2 all wall/gran 100000.0 20000.0 50.0 30.0 0.5 1 zcylinder 15.0 wiggle z 3.0 2.0
```

Description:

Bound the simulation domain of a granular system with a frictional wall. All particles in the group interact with the wall when they are close enough to touch it.

The first set of parameters (Kn, Kt, gamma_n, gamma_t, xmu, and dampflag) have the same meaning as those specified with the pair_style granular force fields. This means a NULL can be used for either Kt or gamma_t as described on that page. If a NULL is used for Kt, then a default value is used where Kt = 2/7 Kn. If a NULL is used for gamma_t, then a default value is used where gamma_t = 1/2 gamma_n.

The nature of the wall/particle interactions are determined by which pair_style is used in your input script: *hooke*, *hooke/history*, or *hertz/history*. The equation for the force between the wall and particles touching it is the same as

the corresponding equation on the pair_style granular doc page, in the limit of one of the two particles going to infinite radius and mass (flat wall). I.e. delta = radius − r = overlap of particle with wall, m_eff = mass of particle, and sqrt(RiRj/Ri+Rj) becomes sqrt(radius of particle). The units for Kn, Kt, gamma_n, and gamma_t are as described on that doc page. The meaning of xmu and dampflag are also as described on that page. Note that you can choose different values for these 6 wall/particle coefficients than for particle/particle interactions, if you wish your wall to interact differently with the particles, e.g. if the wall is a different material.

IMPORTANT NOTE: As discussed on the doc page for pair_style granular, versions of LAMMPS before 9Jan09 used a different equation for Hertzian interactions. This means Hertizian wall/particle interactions have also changed. They now include a sqrt(radius) term which was not present before. Also the previous versions used Kn and Kt from the pairwise interaction and hardwired dampflag to 1, rather than letting them be specified directly. This means you can set the values of the wall/particle coefficients appropriately in the current code to reproduce the results of a prevoius Hertzian monodisperse calculation. For example, for the common case of a monodisperse system with particles of diameter 1, Kn, Kt, gamma_n, and gamma_s should be set sqrt(2.0) larger than they were previously.

The *wallstyle* can be planar or cylindrical. The 3 planar options specify a pair of walls in a dimension. Wall positions are given by *lo* and *hi*. Either of the values can be specified as NULL if a single wall is desired. For a *zcylinder* wallstyle, the cylinder's axis is at x = y = 0.0, and the radius of the cylinder is specified.

Optionally, the wall can be moving, if the *wiggle* or *shear* keywords are appended. Both keywords cannot be used together.

For the *wiggle* keyword, the wall oscillates sinusoidally, similar to the oscillations of particles which can be specified by the fix_move command. This is useful in packing simulations of granular particles. The arguments to the *wiggle* keyword specify a dimension for the motion, as well as it's *amplitude* and *period*. Note that if the dimension is in the plane of the wall, this is effectively a shearing motion. If the dimension is perpendicular to the wall, it is more of a shaking motion. A *zcylinder* wall can only be wiggled in the z dimension.

Each timestep, the position of a wiggled wall in the appropriate *dim* is set according to this equation:

```
position = coord + A − A cos (omega * delta)
```

where *coord* is the specified initial position of the wall, *A* is the *amplitude*, *omega* is 2 PI / *period*, and *delta* is the time elapsed since the fix was specified. The velocity of the wall is set to the derivative of this expression.

For the *shear* keyword, the wall moves continuously in the specified dimension with velocity *vshear*. The dimension must be tangential to walls with a planar *wallstyle*, e.g. in the *y* or *z* directions for an *xplane* wall. For *zcylinder* walls, a dimension of *z* means the cylinder is moving in the z−direction along it's axis. A dimension of *x* or *y* means the cylinder is spinning around the z−axis, either in the clockwise direction for *vshear* > 0 or counter−clockwise for *vshear* < 0. In this case, *vshear* is the tangential velocity of the wall at whatever *radius* has been defined.

Restart, fix_modify, output, run start/stop, minimize info:

This fix writes the shear friction state of atoms interacting with the wall to binary restart files, so that a simulation can continue correctly if granular potentials with shear "history" effects are being used. See the read_restart command for info on how to re−specify a fix in an input script that reads a restart file, so that the operation of the fix continues in an uninterrupted fashion.

None of the fix_modify options are relevant to this fix. No global or per−atom quantities are stored by this fix for access by various output commands. No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions:

This fix is part of the "granular" package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

Any dimension (xyz) that has a granular wall must be non−periodic.

Related commands:

fix_move, pair_style granular

Default: none

fix wall/reflect command

Syntax:

```
fix ID group−ID wall/reflect face arg ... keyword value ...
```

- ID, group−ID are documented in fix command
- wall/reflect = style name of this fix command
- one or more face/arg pairs may be appended
- face = *xlo* or *xhi* or *ylo* or *yhi* or *zlo* or *zhi*

```
xlo,ylo,zlo arg = EDGE or constant or variable
    EDGE = current lo edge of simulation box
    constant = number like 0.0 or −30.0 (distance units)
    variable = equal−style variable like v_x or v_wiggle
  xhi,yhi,zhi arg = EDGE or constant or variable
    EDGE = current hi edge of simulation box
    constant = number like 50.0 or 100.3 (distance units)
    variable = equal−style variable like v_x or v_wiggle
```

- zero or more keyword/value pairs may be appended
- keyword = *units*

```
units value = lattice or box
    lattice = the wall position is defined in lattice units
    box = the wall position is defined in simulation box units
```

Examples:

```
fix xwalls all wall/reflect xlo EDGE xhi EDGE
fix walls all wall/reflect xlo 0.0 ylo 10.0 units box
fix top all wall/reflect zhi v_pressdown
```

Description:

Bound the simulation with one or more walls which reflect particles in the specified group when they attempt to move thru them.

Reflection means that if an atom moves outside the wall on a timestep by a distance delta (e.g. due to fix nve), then it is put back inside the face by the same delta, and the sign of the corresponding component of its velocity is flipped.

When used in conjunction with fix nve and run_style verlet, the resultant time−integration algorithm is equivalent to the primitive splitting algorithm (PSA) described by Bond. Because each reflection event divides the corresponding timestep asymmetrically, energy conservation is only satisfied to O(dt), rather than to O(dt^2) as it would be for velocity−Verlet integration without reflective walls.

Up to 6 walls or faces can be specified in a single command: *xlo*, *xhi*, *ylo*, *yhi*, *zlo*, *zhi*. A *lo* face reflects particles that move to a coordinate less than the wall position, back in the *hi* direction. A *hi* face reflects particles that move to a coordinate higher than the wall position, back in the *lo* direction.

The position of each wall can be specified in one of 3 ways: as the EDGE of the simulation box, as a constant value, or as a variable. If EDGE is used, then the corresponding boundary of the current simulation box is used. If a numeric constant is specified then the wall is placed at that position in the appropriate dimension (x, y, or z). In

both the EDGE and constant cases, the wall will never move. If the wall position is a variable, it should be specified as v_name, where name is an equal−style variable name. In this case the variable is evaluated each timestep and the result becomes the current position of the reflecting wall. Equal−style variables can specify formulas with various mathematical functions, and include thermo_style command keywords for the simulation box parameters and timestep and elapsed time. Thus it is easy to specify a time−dependent wall position.

The *units* keyword determines the meaning of the distance units used to define a wall position, but only when a numeric constant is used. It is not relevant when EDGE or a variable is used to specify a face position.

A *box* value selects standard distance units as defined by the units command, e.g. Angstroms for units = real or metal. A *lattice* value means the distance units are in lattice spacings. The lattice command must have been previously used to define the lattice spacings.

Here are examples of variable definitions that move the wall position in a time−dependent fashion using equal−style variables.

```
variable ramp equal ramp(0,10)
fix 1 all wall xlo v_ramp 1.0 1.0 2.5

variable linear equal vlinear(0,20)
fix 1 all wall xlo v_linear 1.0 1.0 2.5

variable wiggle equal swiggle(0.0,5.0,3.0)
fix 1 all wall xlo v_wiggle 1.0 1.0 2.5

variable wiggle equal cwiggle(0.0,5.0,3.0)
fix 1 all wall xlo v_wiggle 1.0 1.0 2.5
```

The ramp(lo,hi) function adjusts the wall position linearly from lo to hi over the course of a run. The linear(c0,velocity) function does something similar using the equation position = c0 + velocity*delta, where delta is the elapsed time.

The swiggle(c0,A,period) function causes the wall position to oscillate sinusoidally according to this equation, where omega = 2 PI / period:

```
position = c0 + A sin(omega*delta)
```

The cwiggle(c0,A,period) function causes the wall position to oscillate sinusoidally according to this equation, which will have an initial wall velocity of 0.0, and thus may impose a gentler perturbation on the particles:

```
position = c0 + A (1 − cos(omega*delta))
```

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix. No global or per−atom quantities are stored by this fix for access by various output commands. No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions:

Any dimension (xyz) that has a reflecting wall must be non−periodic.

A reflecting wall should not be used with rigid bodies such as those defined by a "fix rigid" command. This is because the wall/reflect displaces atoms directly rather than exerts a force on them. For rigid bodies, use a soft wall instead, such as fix wall/lj93. LAMMPS will flag the use of a rigid fix with fix wall/reflect with a warning, but will not generate an error.

Related commands:

fix wall/lj93 command

Default: none

(Bond) Bond and Leimkuhler, SIAM J Sci Comput, 30, p 134 (2007).

fix wall/region command

Syntax:

```
fix ID group-ID wall/region region-ID style epsilon sigma cutoff
```

- ID, group−ID are documented in fix command
- wall/region = style name of this fix command
- region−ID = region whose boundary will act as wall
- style = *lj93* or *lj126* or *colloid* or *harmonic*
- epsilon = strength factor for wall−particle interaction (energy or energy/distance^2 units)
- sigma = size factor for wall−particle interaction (distance units)
- cutoff = distance from wall at which wall−particle interaction is cut off (distance units)

Examples:

```
fix wall all wall/region mySphere lj93 1.0 1.0 2.5
```

Description:

Treat the surface of the geometric region defined by the *region−ID* as a bounding wall which interacts with nearby particles according to the specified style. The distance between a particle and the surface is the distance to the nearest point on the surface and the force the wall exerts on the particle is along the direction between that point and the particle, which is the direction normal to the surface at that point.

Regions are defined using the region command. Note that the region volume can be interior or exterior to the bounding surface, which will determine in which direction the surface interacts with particles, i.e. the direction of the surface normal. Regions can either be primitive shapes (block, sphere, cylinder, etc) or combinations of primitive shapes specified via the *union* or *intersect* region styles. These latter styles can be used to construct particle containers with complex shapes. Regions can also change over time via keywords like *linear*, *wiggle*, and *rotate*, which when used with this fix, have the effect of moving the region surface in a prescribed manner.

IMPORTANT NOTE: As discussed on the region command doc page, regions in LAMMPS do not get wrapped across periodic boundaries. It is up to you to insure that periodic or non−periodic boundaries are specified appropriately via the boundary command when using a region as a wall that bounds particle motion. This also means that if you embed a region in your simulation box and want it to repulse particles from its surface (using the "side out" option in the region command), that its repulsive force will not be felt across a periodic boundary.

IMPORTANT NOTE: For primitive regions with sharp corners and/or edges (e.g. a block or cylinder), wall/particle forces are computed accurately for both interior and exterior regions. For *union* and *intersect* regions, additional sharp corners and edges may be present due to the intersection of the surfaces of 2 or more primitive volumes. These corners and edges can be of two types: concave or convex. Concave points/edges are like the corners of a cube as seen by particles in the interior of a cube. Wall/particle forces around these features are computed correctly. Convex points/edges are like the corners of a cube as seen by particles exterior to the cube, i.e. the points jut into the volume where particles are present. LAMMPS does NOT compute the location of these convex points directly, and hence wall/particle forces in the cutoff volume around these points suffer from inaccuracies. The basic problem is that the outward normal of the surface is not continuous at these points. This can cause particles to feel no force (they don't "see" the wall) when in one location, then move a distance epsilon, and suddenly feel a large force because they now "see" the wall. In the worst−case scenario, this can blow particles out of the simulation box. Thus, as a general rule you should not use the fix wall/region command with

union or *interesect* regions that have convex points or edges.

The energy of wall−particle interactions depends on the specified style.

For style *lj93*, the energy E is given by the 9/3 potential:

$$E = \epsilon \left[\frac{2}{15} \left(\frac{\sigma}{r} \right)^9 - \left(\frac{\sigma}{r} \right)^3 \right] \qquad r < r_c$$

For style *lj126*, the energy E is given by the 12/6 potential:

$$E = 4\epsilon \left[\left(\frac{\sigma}{r} \right)^{12} - \left(\frac{\sigma}{r} \right)^6 \right] \qquad r < r_c$$

For style *colloid*, the energy E is given by an integrated form of the pair_style colloid potential:

$$\begin{aligned} E &= 144\phi^2\epsilon \left[\frac{\sigma^6}{7560} \left(\frac{6R-D}{D^7} + \frac{D+8R}{(D+2R)^7} \right) - \right. \\ &\left. \frac{1}{6} \left(\frac{2R(D+R) + D(D+2R)\left[\ln D - \ln(D+2R)\right]}{D(D+2R)} \right) \right] \qquad r < r_c \end{aligned}$$

For style *wall/harmonic*, the energy E is given by a harmonic spring potential:

$$E = \epsilon \left(r - r_c \right)^2 \qquad r < r_c$$

In all cases, *r* is the distance from the particle to the region surface, and Rc is the *cutoff* distance at which the particle and surface no longer interact. The energy of the wall potential is shifted so that the wall−particle interaction energy is 0.0 at the cutoff distance.

For the *lj93* and *lj126* styles, *epsilon* and *sigma* are the usual Lennard−Jones parameters, which determine the strength and size of the particle as it interacts with the wall. Epsilon has energy units. Note that this *epsilon* and *sigma* may be different than any *epsilon* or *sigma* values defined for a pair style that computes particle−particle interactions.

The *lj93* interaction is derived by integrating over a 3d half−lattice of Lennard−Jones 12/6 particles. The *lj126* interaction is effectively a harder, more repulsive wall interaction.

For the *colloid* style, *epsilon* is effectively a Hamaker constant with energy units for the colloid−wall interaction, *R* is the radius of the colloid particle, *D* is the distance from the surface of the colloid particle to the wall (r−R), and *sigma* is the size of a constituent LJ particle inside the colloid particle. Note that the cutoff distance Rc in this case is the distance from the colloid particle center to the wall.

The *colloid* interaction is derived by integrating over constituent LJ particles of size *sigma* within the colloid particle and a 3d half−lattice of Lennard−Jones 12/6 particles of size *sigma* in the wall.

For the *wall/harmonic* style, *epsilon* is effectively the spring constant K, and has units (energy/distance^2). The input parameter *sigma* is ignored. The minimum energy position of the harmonic spring is at the *cutoff*. This is a repulsive−only spring since the interaction is truncated at the *cutoff*

IMPORTANT NOTE: For all of the styles, you must insure that r is always > 0 for all particles in the group, or LAMMPS will generate an error. This means you cannot start your simulation with particles on the region surface (r = 0) or with particles on the wrong side of the region surface (r < 0). For the *wall/lj93* and *wall/lj126* styles, the energy of the wall/particle interaction (and hence the force on the particle) blows up as r −> 0. The *wall/colloid* style is even more restrictive, since the energy blows up as D = r−R −> 0. This means the finite−size particles of radius R must be a distance larger than R from the region surface. The *harmonic* style is a softer potential and does not blow up as r −> 0, but you must use a large enough *epsilon* that particles always reamin on the correct side of the region surface (r > 0).

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files.

The fix_modify *energy* option is supported by this fix to add the energy of interaction between atoms and the wall to the system's potential energy as part of thermodynamic output.

This fix computes a global scalar energy and a global 3−length vector of forces, which can be accessed by various output commands. The scalar energy is the sum of energy interactions for all particles interacting with the wall represented by the region surface. The 3 vector quantities are the x,y,z components of the total force acting on the wall due to the particles. The scalar and vector values calculated by this fix are "extensive".

No parameter of this fix can be used with the *start/stop* keywords of the run command.

The forces due to this fix are imposed during an energy minimization, invoked by the minimize command.

IMPORTANT NOTE: If you want the atom/wall interaction energy to be included in the total potential energy of the system (the quantity being minimized), you MUST enable the fix_modify *energy* option for this fix.

Restrictions: none

Related commands:

fix wall/lj93, fix wall/lj126, fix wall/colloid, fix wall/gran

Default: none

fix wall/srd command

Syntax:

```
fix ID group-ID wall/srd face arg ... keyword value ...
```

- ID, group−ID are documented in fix command
- wall/srd = style name of this fix command
- one or more face/arg pairs may be appended
- face = *xlo* or *xhi* or *ylo* or *yhi* or *zlo* or *zhi*

```
xlo,ylo,zlo arg = EDGE or constant or variable
    EDGE = current lo edge of simulation box
    constant = number like 0.0 or -30.0 (distance units)
    variable = equal-style variable like v_x or v_wiggle
  xhi,yhi,zhi arg = EDGE or constant or variable
    EDGE = current hi edge of simulation box
    constant = number like 50.0 or 100.3 (distance units)
    variable = equal-style variable like v_x or v_wiggle
```

- zero or more keyword/value pairs may be appended
- keyword = *units*

```
units value = lattice or box
    lattice = the wall position is defined in lattice units
    box = the wall position is defined in simulation box units
```

Examples:

```
fix xwalls all wall/srd xlo EDGE xhi EDGE
fix walls all wall/srd xlo 0.0 ylo 10.0 units box
fix top all wall/srd zhi v_pressdown
```

Description:

Bound the simulation with one or more walls which interact with stochastic reaction dynamics (SRD) particles as slip (smooth) or no−slip (rough) flat surfaces. The wall interaction is actually invoked via the fix srd command, only on the group of SRD particles it defines, so the group setting for the fix wall/srd command is ignored.

A particle/wall collision occurs if an SRD particle moves outside the wall on a timestep. This alters the position and velocity of the SRD particle and imparts a force to the wall.

The *collision* and *Tsrd* settings specified via the fix srd command affect the SRD/wall collisions. A *slip* setting for the *collision* keyword means that the tangential component of the SRD particle momentum is preserved. Thus only a normal force is imparted to the wall. The normal component of the new SRD velocity is sampled from a Gaussian distribution at temperature *Tsrd*.

For a *noslip* setting of the *collision* keyword, both the normal and tangential components of the new SRD velocity are sampled from a Gaussian distribution at temperature *Tsrd*. Additionally, a new tangential direction for the SRD velocity is chosen randomly. This collision style imparts both a normal and tangential force to the wall.

Up to 6 walls or faces can be specified in a single command: *xlo*, *xhi*, *ylo*, *yhi*, *zlo*, *zhi*. A *lo* face reflects particles that move to a coordinate less than the wall position, back in the *hi* direction. A *hi* face reflects particles that move to a coordinate higher than the wall position, back in the *lo* direction.

The position of each wall can be specified in one of 3 ways: as the EDGE of the simulation box, as a constant value, or as a variable. If EDGE is used, then the corresponding boundary of the current simulation box is used. If a numeric constant is specified then the wall is placed at that position in the appropriate dimension (x, y, or z). In both the EDGE and constant cases, the wall will never move. If the wall position is a variable, it should be specified as v_name, where name is an equal−style variable name. In this case the variable is evaluated each timestep and the result becomes the current position of the reflecting wall. Equal−style variables can specify formulas with various mathematical functions, and include thermo_style command keywords for the simulation box parameters and timestep and elapsed time. Thus it is easy to specify a time−dependent wall position.

IMPORTANT NOTE: Because the trajectory of the SRD particle is tracked as it collides with the wall, you must insure that r = distance of the particle from the wall, is always > 0 for SRD particles, or LAMMPS will generate an error. This means you cannot start your simulation with SRD particles at the wall position *coord* (r = 0) or with particles on the wrong side of the wall (r < 0).

IMPORTANT NOTE: If you have 2 or more walls that come together at an edge or corner (e.g. walls in the x and y dimensions), then be sure to set the *overlap* keyword to *yes* in the fix srd command, since the walls effectively overlap when SRD particles collide with them. LAMMPS will issue a warning if you do not do this.

IMPORTANT NOTE: The walls of this fix only interact with SRD particles, as defined by the fix srd command. If you are simulating a mixture containing other kinds of particles, then you should typically use another wall command to act on the other particles. Since SRD particles will be colliding both with the walls and the other particles, it is important to insure that the other particle's finite extent does not overlap an SRD wall. If you do not do this, you may generate errors when SRD particles end up "inside" another particle or a wall at the beginning of a collision step.

The *units* keyword determines the meaning of the distance units used to define a wall position, but only when a numeric constant is used. It is not relevant when EDGE or a variable is used to specify a face position.

A *box* value selects standard distance units as defined by the units command, e.g. Angstroms for units = real or metal. A *lattice* value means the distance units are in lattice spacings. The lattice command must have been previously used to define the lattice spacings.

Here are examples of variable definitions that move the wall position in a time−dependent fashion using equal−style variables.

```
variable ramp equal ramp(0,10)
fix 1 all wall xlo v_ramp 1.0 1.0 2.5

variable linear equal vlinear(0,20)
fix 1 all wall xlo v_linear 1.0 1.0 2.5

variable wiggle equal swiggle(0.0,5.0,3.0)
fix 1 all wall xlo v_wiggle 1.0 1.0 2.5

variable wiggle equal cwiggle(0.0,5.0,3.0)
fix 1 all wall xlo v_wiggle 1.0 1.0 2.5
```

The ramp(lo,hi) function adjusts the wall position linearly from lo to hi over the course of a run. The linear(c0,velocity) function does something similar using the equation position = c0 + velocity*delta, where delta is the elapsed time.

The swiggle(c0,A,period) function causes the wall position to oscillate sinusoidally according to this equation, where omega = 2 PI / period:

```
position = c0 + A sin(omega*delta)
```

The cwiggle(c0,A,period) function causes the wall position to oscillate sinusoidally according to this equation, which will have an initial wall velocity of 0.0, and thus may impose a gentler perturbation on the particles:

```
position = c0 + A (1 − cos(omega*delta))
```

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix.

This fix computes a global array of values which can be accessed by various output commands. The number of rows in the array is equal to the number of walls defined by the fix. The number of columns is 3, for the x,y,z components of force on each wall.

Note that an outward normal force on a wall will be a negative value for *lo* walls and a positive value for *hi* walls. The array values calculated by this fix are "extensive".

No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions:

Any dimension (xyz) that has an SRD wall must be non−periodic.

Related commands:

fix srd

Default: none

group command

Syntax:

```
group ID style args
```

- ID = user−defined name of the group
- style = *delete* or *region* or *type* or *id* or *molecule* or *subtract* or *union* or *intersect*

```
  delete = no args
    region args = region-ID
    type or id or molecule
      args = one or more atom types, atom IDs, or molecule IDs
      args = logical value
        logical = "" or ">=" or "==" or "!="
        value = an atom type or atom ID or molecule ID (depending on style)
      args = logical value1 value2
        logical = ""
        value1,value2 = atom types or atom IDs or molecule IDs
          (depending on      style)
    subtract args = two or more group IDs
    union args = one or more group IDs
    intersect args = two or more group IDs
```

Examples:

```
group edge region regstrip
group water type 3 4
group sub id <= 150
group polyA molecule  50 250
group boundary subtract all a2 a3
group boundary union lower upper
group boundary intersect upper flow
group boundary delete
```

Description:

Identify a collection of atoms as belonging to a group. The group ID can then be used in other commands such as fix, compute, dump, or velocity to act on those atoms together.

If the group ID already exists, the group command adds the specified atoms to the group.

The *delete* style removes the named group and un−assigns all atoms that were assigned to that group. Since there is a restriction (see below) that no more than 32 groups can be defined at any time, the *delete* style allows you to remove groups that are no longer needed, so that more can be specified. You cannot delete a group if it has been used to define a current fix or compute or dump.

The *region* style puts all atoms in the region volume into the group. Note that this is a static one−time assignment. The atoms remain assigned (or not assigned) to the group even in they later move out of the region volume.

The *type*, *id*, and *molecule* styles put all atoms with the specified atom types, atom IDs, or molecule IDs into the group. These 3 styles can have their arguments specified in one of two formats. The 1st format is a list of values (types or IDs). For example, the 2nd command in the examples above puts all atoms of type 3 or 4 into the group named *water*. The 2nd format is a *logical* followed by one or two values (type or ID). The 7 valid logicals are

listed above. All the logicals except take a single argument. The 3rd example above adds all atoms with IDs from 1 to 150 to the group named *sub*. The logical means "between" and takes 2 arguments. The 4th example above adds all atoms belonging to molecules with IDs from 50 to 250 (inclusive) to the group named polyA.

The *subtract* style takes a list of two or more existing group names as arguments. All atoms that belong to the 1st group, but not to any of the other groups are added to the specified group.

The *union* style takes a list of one or more existing group names as arguments. All atoms that belong to any of the listed groups are added to the specified group.

The *intersect* style takes a list of two or more existing group names as arguments. Atoms that belong to every one of the listed groups are added to the specified group.

A group with the ID *all* is predefined. All atoms belong to this group. This group cannot be deleted.

Restrictions:

There can be no more than 32 groups defined at one time, including "all".

Related commands:

dump, fix, region, velocity

Default:

All atoms belong to the "all" group.

if command

Syntax:

```
if boolean then t1 t2 ... elif boolean f1 f2 ... elif boolean f1 f2 ... else e1 e2 ...
```

- boolean = a Boolean expression evaluated as TRUE or FALSE (see below)
- then = required word
- t1,t2,...,tN = one or more LAMMPS commands to execute if condition is met, each enclosed in quotes
- elif = optional word, can appear multiple times
- f1,f2,...,fN = one or more LAMMPS commands to execute if elif condition is met, each enclosed in quotes (optional arguments)
- else = optional argument
- e1,e2,...,eN = one or more LAMMPS commands to execute if no condition is met, each enclosed in quotes (optional arguments)

Examples:

```
if "${steps} > 1000" then exit
if "$x <= $y" then "print X is smaller = $x" else "print Y is smaller = $y"
if "(${eng} > 0.0) || ($n <1000)" then &  "timestep 0.005" &elif $n ${eng_previous}" then "jump file
```

Description:

This command provides an in−then−else capability within an input script. A Boolean expression is evaluted and the result is TRUE or FALSE. Note that as in the examples above, the expression can contain variables, as defined by the variable command, which will be evaluated as part of the expression. Thus a user−defined formula that reflects the current state of the simulation can be used to issue one or more new commands.

If the result of the Boolean expression is TRUE, then one or more commands (t1, t2, ..., tN) are executed. If it is FALSE, then Boolean expressions associated with successive elif keywords are evaluated until one is found to be true, in which case its commands (f1, f2, ..., fN) are executed. If no Boolean expression is TRUE, then the commands associated witht the else keyword, namely (e1, e2, ..., eN), are executed. The elif and else keywords and their associated commands are optional. If they aren't specified and the initial Boolean expression is FALSE, then no commands are executed.

The syntax for Boolean expressions is described below.

Each command (t1, f1, e1, etc) can be any valid LAMMPS input script command. If the command is more than one word, it must enclosed in quotes, so it will be treated as a single argument, as in the examples above.

IMPORTANT NOTE: If a command itself requires a quoted argument (e.g. a print command), then double and single quotes can be used and nested in the usual manner, as in the examples above and below. See this section of the manual for more details on using quotes in arguments. Only one of level of nesting is allowed, but that should be sufficient for most use cases.

Note that by using the line continuation character "the if command can be spread across many lines, though it is still a single command:

```
if "$a <$b" then &  "print 'Minimum value = $a'" &  "run 1000" &else &
  'print "Minimum value = $b"' &  "minimize 0.001 0.001 1000 10000"
```

Note that if one of the commands to execute is an invalid LAMMPS command, such as "exit" in the first example above, then executing the command will cause LAMMPS to halt.

Note that by jumping to a label in the same input script, the if command can be used to break out of a loop. See the variable delete command for info on how to delete the associated loop variable, so that it can be re−used later in the input script.

Here is an example of a double loop which uses the if and jump commands to break out of the inner loop when a condition is met, then continues iterating thru the outer loop.

```
label       loopa
variable    a loop 5
  label     loopb
  variable  b loop 5
  print     "A,B = $a,$b"
  run       10000
  if        '$b > 2' then "print 'Jumping to another script'" "jump in.script break"
  next      b
  jump      in.script loopb
label       break
variable    b delete

next        a
jump        in.script loopa
```

The Boolean expressions for the if and elif keywords have a C−like syntax. Note that each expression is a single argument within the if command. Thus if you want to include spaces in the expression for clarity, you must enclose the entire expression in quotes.

An expression is built out of numbers:

```
0.2, 100, 1.0e20, −15.4, etc
```

and Boolean operators:

```
A == B, A != B, A <B, A  <= B, A > B, A >= B, A &B, A || B, !A
```

Each A and B is a number or a variable reference like $a or ${abc}, or another Boolean expression.

If a variable is used it must produce a number when evaluated and substituted for in the expression, else an error will be generated.

Expressions are evaluated left to right and have the usual C−style precedence: the unary logical NOT operator "!" has the highest precedence, the 4 relational operators "", and ">=" are next; the two remaining relational operators "==" and "!=" are next; then the logical AND operator " and finally the logical OR operator "||" has the lowest precedence. Parenthesis can be used to group one or more portions of an expression and/or enforce a different order of evaluation than what would occur with the default precedence.

The 6 relational operators return either a 1.0 or 0.0 depending on whether the relationship between x and y is TRUE or FALSE. The logical AND operator will return 1.0 if both its arguments are non−zero, else it returns 0.0. The logical OR operator will return 1.0 if either of its arguments is non−zero, else it returns 0.0. The logical NOT operator returns 1.0 if its argument is 0.0, else it returns 0.0.

The overall Boolean expression produces a TRUE result if the result is non−zero. If the result is zero, the expression result is FALSE.

Restrictions: none

Related commands:

variable, print

Default: none

improper_style class2 command

Syntax:

```
improper_style class2
```

Examples:

```
improper_style class2
improper_coeff 1 100.0 0
improper_coeff * aa 0.0 0.0 0.0 115.06 130.01 115.06
```

Description:

The *class2* improper style uses the potential

$$
\begin{aligned}
E &= E_i + E_{aa} \\
E_i &= K[\frac{\chi_{ijkl} + \chi_{kjli} + \chi_{ljik}}{3} - \chi_0]^2 \\
E_{aa} &= M_1(\theta_{ijk} - \theta_1)(\theta_{kjl} - \theta_3) + \\
&\quad M_2(\theta_{ijk} - \theta_1)(\theta_{ijl} - \theta_2) + \\
&\quad M_3(\theta_{ijl} - \theta_2)(\theta_{kjl} - \theta_3)
\end{aligned}
$$

where Ei is the improper term and Eaa is an angle−angle term. The 3 X terms in Ei are an average over 3 out−of−plane angles.

The 4 atoms in an improper quadruplet (listed in the data file read by the read_data command) are ordered I,J,K,L. X_IJKL refers to the angle between the plane of I,J,K and the plane of J,K,L, and the bond JK lies in both planes. Similarly for X_KJLI and X_LJIK. Note that atom J appears in the common bonds (JI, JK, JL) of all 3 X terms. Thus J (the 2nd atom in the quadruplet) is the atom of symmetry in the 3 X angles.

The subscripts on the various theta's refer to different combinations of 3 atoms (I,J,K,L) used to form a particular angle. E.g. Theta_IJL is the angle formed by atoms I,J,L with J in the middle. Theta1, theta2, theta3 are the equilibrium positions of those angles. Again, atom J (the 2nd atom in the quadruplet) is the atom of symmetry in the theta angles, since it is always the center atom.

Since atom J is the atom of symmetry, normally the bonds J−I, J−K, J−L would exist for an improper to be defined between the 4 atoms, but this is not required.

See (Sun) for a description of the COMPASS class2 force field.

Coefficients for the Ei and Eaa formulas must be defined for each improper type via the improper_coeff command as in the example above, or in the data file or restart files read by the read_data or read_restart commands.

These are the 2 coefficients for the Ei formula:

- K (energy/radian^2)
- X0 (degrees)

X0 is specified in degrees, but LAMMPS converts it to radians internally; hence the units of K are in energy/radian^2.

For the Eaa formula, each line in a improper_coeff command in the input script lists 7 coefficients, the first of which is "aa" to indicate they are AngleAngle coefficients. In a data file, these coefficients should be listed under a "AngleAngle Coeffs" heading and you must leave out the "aa", i.e. only list 6 coefficients after the improper type.

- aa
- M1 (energy/distance)
- M2 (energy/distance)
- M3 (energy/distance)
- theta1 (degrees)
- theta2 (degrees)
- theta3 (degrees)

The theta values are specified in degrees, but LAMMPS converts them to radians internally; hence the units of M are in energy/radian^2.

Restrictions:

This improper style can only be used if LAMMPS was built with the "class2" package. See the Making LAMMPS section for more info on packages.

Related commands:

improper_coeff

Default: none

(Sun) Sun, J Phys Chem B 102, 7338−7364 (1998).

improper_coeff command

Syntax:

```
improper_coeff N args
```

- N = improper type (see asterisk form below)
- args = coefficients for one or more improper types

Examples:

```
improper_coeff 1 300.0 0.0
improper_coeff * 80.2 −1 2
improper_coeff *4 80.2 −1 2
```

Description:

Specify the improper force field coefficients for one or more improper types. The number and meaning of the coefficients depends on the improper style. Improper coefficients can also be set in the data file read by the read_data command or in a restart file.

N can be specified in one of two ways. An explicit numeric value can be used, as in the 1st example above. Or a wild−card asterisk can be used to set the coefficients for multiple improper types. This takes the form "*" or "*n" or "n*" or "m*n". If N = the number of improper types, then an asterisk with no numeric values means all types from 1 to N. A leading asterisk means all types from 1 to n (inclusive). A trailing asterisk means all types from n to N (inclusive). A middle asterisk means all types from m to n (inclusive).

Note that using an improper_coeff command can override a previous setting for the same improper type. For example, these commands set the coeffs for all improper types, then overwrite the coeffs for just improper type 2:

```
improper_coeff * 300.0 0.0
improper_coeff 2 50.0 0.0
```

A line in a data file that specifies improper coefficients uses the exact same format as the arguments of the improper_coeff command in an input script, except that wild−card asterisks should not be used since coefficients for all N types must be listed in the file. For example, under the "Improper Coeffs" section of a data file, the line that corresponds to the 1st example above would be listed as

```
1 300.0 0.0
```

The improper_style class2 is an exception to this rule, in that an additional argument is used in the input script to allow specification of the cross−term coefficients. See its doc page for details.

Here is an alphabetic list of improper styles defined in LAMMPS. Click on the style to display the formula it computes and coefficients specified by the associated improper_coeff command:

- improper_style none − turn off improper interactions
- improper_style hybrid − define multiple styles of improper interactions

- improper_style class2 − COMPASS (class 2) improper
- improper_style cvff − CVFF improper

- improper_style harmonic − harmonic improper
- improper_style umbrella − DREIDING improper

There are also additional improper styles submitted by users which are included in the LAMMPS distribution. The list of these with links to the individual styles are given in the improper section of this page.

Restrictions:

This command must come after the simulation box is defined by a read_data, read_restart, or create_box command.

An improper style must be defined before any improper coefficients are set, either in the input script or in a data file.

Related commands:

improper_style

Default: none

improper_style cvff command

Syntax:

```
improper_style cvff
```

Examples:

```
improper_style cvff
improper_coeff 1 80.0 -1 4
```

Description:

The *cvff* improper style uses the potential

$$E = K[1 + d\cos(n\phi)]$$

where phi is the Wilson out−of−plane angle.

If the 4 atoms in an improper quadruplet (listed in the data file read by the read_data command) are ordered I,J,K,L then the Wilson angle is between the plane of I,J,K and the plane of J,K,L. This is essentially a dihedral angle, which is why the formula for this improper style is the same as for dihedral_style harmonic. Alternatively, you can think of atoms J,K,L as being in a plane, and atom I above the plane, and the Wilson angle as a measure of how far out−of−plane I is with respect to the other 3 atoms.

Note that defining 4 atoms to interact in this way, does not mean that bonds necessarily exist between I−J, J−K, or K−L, as they would in a linear dihedral. Normally, the bonds I−J, I−K, I−L would exist for an improper to be defined between the 4 atoms.

The following coefficients must be defined for each improper type via the improper_coeff command as in the example above, or in the data file or restart files read by the read_data or read_restart commands:

- K (energy)
- d (+1 or −1)
- n (0,1,2,3,4,6)

Restrictions:

This improper style can only be used if LAMMPS was built with the "molecular" package (which it is by default). See the Making LAMMPS section for more info on packages.

Related commands:

improper_coeff

Default: none

improper_style harmonic command

Syntax:

```
improper_style harmonic
```

Examples:

```
improper_style harmonic
improper_coeff 1 100.0 0
```

Description:

The *harmonic* improper style uses the potential

$$E = K(\chi - \chi_0)^2$$

where X is the improper angle, X0 is its equilibrium value, and K is a prefactor. Note that the usual 1/2 factor is included in K.

If the 4 atoms in an improper quadruplet (listed in the data file read by the read_data command) are ordered I,J,K,L then X is the angle between the plane of I,J,K and the plane of J,K,L. Alternatively, you can think of atoms J,K,L as being in a plane, and atom I above the plane, and X as a measure of how far out−of−plane I is with respect to the other 3 atoms.

Note that defining 4 atoms to interact in this way, does not mean that bonds necessarily exist between I−J, J−K, or K−L, as they would in a linear dihedral. Normally, the bonds I−J, I−K, I−L would exist for an improper to be defined between the 4 atoms.

The following coefficients must be defined for each improper type via the improper_coeff command as in the example above, or in the data file or restart files read by the read_data or read_restart commands:

- K (energy/radian^2)
- X0 (degrees)

X0 is specified in degrees, but LAMMPS converts it to radians internally; hence the units of K are in energy/radian^2.

Restrictions:

This improper style can only be used if LAMMPS was built with the "molecular" package (which it is by default). See the Making LAMMPS section for more info on packages.

Related commands:

improper_coeff

Default: none

improper_style hybrid command

Syntax:

```
improper_style hybrid style1 style2 ...
```

- style1,style2 = list of one or more improper styles

Examples:

```
improper_style hybrid harmonic helix
improper_coeff 1 harmonic 120.0 30
improper_coeff 2 cvff 20.0 −1 2
```

Description:

The *hybrid* style enables the use of multiple improper styles in one simulation. An improper style is assigned to each improper type. For example, impropers in a polymer flow (of improper type 1) could be computed with a *harmonic* potential and impropers in the wall boundary (of improper type 2) could be computed with a *cvff* potential. The assignment of improper type to style is made via the improper_coeff command or in the data file.

In the improper_coeff command, the first coefficient sets the improper style and the remaining coefficients are those appropriate to that style. In the example above, the 2 improper_coeff commands would set impropers of improper type 1 to be computed with a *harmonic* potential with coefficients 120.0, 30 for K, X0. Improper type 2 would be computed with a *cvff* potential with coefficients 20.0, −1, 2 for K, d, n.

If the improper *class2* potential is one of the hybrid styles, it requires additional AngleAngle coefficients be specified in the data file. These lines must also have an additional "class2" argument added after the improper type. For improper types which are assigned to other hybrid styles, use the style name (e.g. "harmonic") appropriate to that style. The AngleAngle coeffs for that improper type will then be ignored.

An improper style of *none* can be specified as the 2nd argument to the improper_coeff command, if you desire to turn off certain improper types.

Restrictions:

This improper style can only be used if LAMMPS was built with the "molecular" package (which it is by default). See the Making LAMMPS section for more info on packages.

Unlike other improper styles, the hybrid improper style does not store improper coefficient info for individual sub−styles in a binary restart files. Thus when retarting a simulation from a restart file, you need to re−specify improper_coeff commands.

Related commands:

improper_coeff

Default: none

improper_style none command

Syntax:

```
improper_style none
```

Examples:

```
improper_style none
```

Description:

Using an improper style of none means improper forces are not computed, even if quadruplets of improper atoms were listed in the data file read by the read_data command.

Restrictions: none

Related commands: none

Default: none

improper_style command

Syntax:

```
improper_style style
```

- style = *none* or *hybrid* or *class2* or *cvff* or *harmonic*

Examples:

```
improper_style harmonic
improper_style cvff
improper_style hybrid cvff harmonic
```

Description:

Set the formula(s) LAMMPS uses to compute improper interactions between quadruplets of atoms, which remain in force for the duration of the simulation. The list of improper quadruplets is read in by a read_data or read_restart command from a data or restart file. Note that the ordering of the 4 atoms in an improper quadruplet determines the the definition of the improper angle used in the formula for each style. See the doc pages of individual styles for details.

Hybrid models where impropers are computed using different improper potentials can be setup using the *hybrid* improper style.

The coefficients associated with an improper style can be specified in a data or restart file or via the improper_coeff command.

All improper potentials store their coefficient data in binary restart files which means improper_style and improper_coeff commands do not need to be re−specified in an input script that restarts a simulation. See the read_restart command for details on how to do this. The one exception is that improper_style *hybrid* only stores the list of sub−styles in the restart file; improper coefficients need to be re−specified.

IMPORTANT NOTE: When both an improper and pair style is defined, the special_bonds command often needs to be used to turn off (or weight) the pairwise interaction that would otherwise exist between a group of 4 bonded atoms.

Here is an alphabetic list of improper styles defined in LAMMPS. Click on the style to display the formula it computes and coefficients specified by the associated improper_coeff command:

- improper_style none − turn off improper interactions
- improper_style hybrid − define multiple styles of improper interactions

- improper_style class2 − COMPASS (class 2) improper
- improper_style cvff − CVFF improper
- improper_style harmonic − harmonic improper
- improper_style umbrella − DREIDING improper

There are also additional improper styles submitted by users which are included in the LAMMPS distribution. The list of these with links to the individual styles are given in the improper section of this page.

Restrictions:

Improper styles can only be set for atom_style choices that allow impropers to be defined.

Most improper styles are part of the "molecular" package. They are only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info on packages. The doc pages for individual improper potentials tell if it is part of a package.

Related commands:

improper_coeff

Default:

```
improper_style none
```

improper_style umbrella command

Syntax:

```
improper_style umbrella
```

Examples:

```
improper_style umbrella
improper_coeff 1 100.0 180.0
```

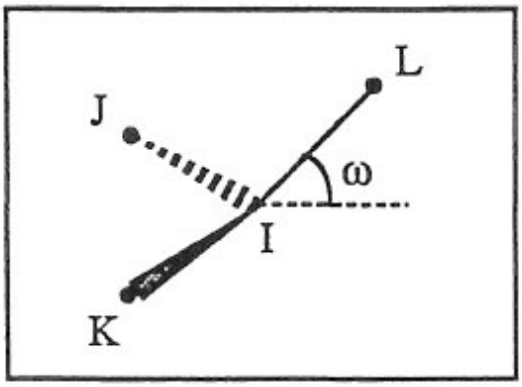
Description:

The *umbrella* improper style uses the following potential, which is commonly referred to as a classic inversion and used in the DREIDING force field:

$$E = \frac{1}{2}K\left(\frac{1+cos\omega_0}{sin\omega_0}\right)^2(cos\omega - cos\omega_0) \qquad \omega_0 \neq 0^o$$

$$E = K\,(1-cos\omega) \qquad \omega_0 = 0^o$$

where K is the force constant and omega is the angle between the IL axis and the IJK plane:



If omega0 = 0 the potential term has a minimum for the planar structure. Otherwise it has two minima at +/− omega0, with a barrier in between.

See (Mayo) for a description of the DREIDING force field.

The following coefficients must be defined for each improper type via the improper_coeff command as in the example above, or in the data file or restart files read by the read_data or read_restart commands:

- K (energy)
- omega0 (degrees)

Restrictions:

This improper style can only be used if LAMMPS was built with the "molecular" package (which it is by default). See the Making LAMMPS section for more info on packages.

Related commands:

improper_coeff

Default: none

(Mayo) Mayo, Olfason, Goddard III, J Phys Chem, 94, 8897−8909 (1990),

include command

Syntax:

```
include file
```

- file = filename of new input script to switch to

Examples:

```
include newfile
include in.run2
```

Description:

This command opens a new input script file and begins reading LAMMPS commands from that file. When the new file is finished, the original file is returned to. Include files can be nested as deeply as desired. If input script A includes script B, and B includes A, then LAMMPS could run for a long time.

If the filename is a variable (see the variable command), different processor partitions can run different input scripts.

Restrictions: none

Related commands:

variable, jump

Default: none

jump command

Syntax:

```
jump file label
```

- file = filename of new input script to switch to
- label = optional label within file to jump to

Examples:

```
jump newfile
jump in.run2 runloop
jump SELF runloop
```

Description:

This command closes the current input script file, opens the file with the specified name, and begins reading LAMMPS commands from that file. Unlike the include command, the original file is not returned to, although by using multiple jump commands it is possible to chain from file to file or back to the original file.

If the word "SELF" is used for the filename, then the current input script is re−opened and read again.

IMPORTANT NOTE: The SELF option is not guaranteed to work when the current input script is being read through stdin (standard input), e.g.

```
lmp_g++ <in.script
```

since the SELF option invokes the C−library rewind() call, which may not be supported for stdin on some systems. This can be worked around by using the −in command−line argument or the −var command−line argument to pass the script name as a variable to the input script In the latter case, the "fname" variable could be used in place of SELF. E.g.

```
lmp_g++ −in in.script

lmp_g++ −var fname n.script <in.script
```

The 2nd argument to the jump command is optional. If specified, it is treated as a label and the new file is scanned (without executing commands) until the label is found, and commands are executed from that point forward. This can be used to loop over a portion of the input script, as in this example. These commands perform 10 runs, each of 10000 steps, and create 10 dump files named file.1, file.2, etc. The next command is used to exit the loop after 10 iterations. When the "a" variable has been incremented for the tenth time, it will cause the next jump command to be skipped.

```
variable a loop 10
label loop
dump 1 all atom 100 file.$a
run 10000
undump 1
next a
jump in.lj loop
```

If the jump *file* argument is a variable, the jump command can be used to cause different processor partitions to run different input scripts. In this example, LAMMPS is run on 40 processors, with 4 partitions of 10 procs each. An in.file containing the example variable and jump command will cause each partition to run a different simulation.

```
mpirun −np 40 lmp_ibm −partition 4x10 −in in.file

variable f world script.1 script.2 script.3 script.4
jump $f
```

Here is an example of a double loop which uses the if and jump commands to break out of the inner loop when a condition is met, then continues iterating thru the outer loop.

```
label       loopa
variable    a loop 5
  label     loopb
  variable  b loop 5
  print     "A,B = $a,$b"
  run       10000
  if        $b > 2 then "jump in.script break"
  next      b
  jump      in.script loopb
label       break
variable    b delete

next        a
jump        in.script loopa
```

Restrictions:

If you jump to a file and it does not contain the specified label, LAMMPS will come to the end of the file and exit.

Related commands:

variable, include, label, next

Default: none

kspace_modify command

Syntax:

```
kspace_modify keyword value ...
```

- one or more keyword/value pairs may be listed
- keyword = *mesh* or *order* or *gewald* or *slab*

```
mesh value = x y z
    x,y,z = PPPM FFT grid size in each dimension
  order value = N
    N = grid extent of Gaussian for PPPM mapping of each charge
  gewald value = r
    r = PPPM G−ewald parameter
  slab value = volfactor
    volfactor = ratio of the total extended volume used in the
      2d approximation compared with the volume of the simulation domain
```

Examples:

```
kspace_modify mesh 24 24 30 order 6
kspace_modify slab 3.0
```

Description:

Set parameters used by the kspace solvers defined by the kspace_style command. Not all parameters are relevant to all kspace styles.

The *mesh* keyword sets the 3d FFT grid size for kspace style pppm. Each dimension must be factorizable into powers of 2, 3, and 5. When this option is not set, the PPPM solver chooses its own grid size, consistent with the user−specified accuracy and pairwise cutoff. Values for x,y,z of 0,0,0 unset the option.

The *order* keyword determines how many grid spacings an atom's charge extends when it is mapped to the FFT grid in kspace style pppm. The default for this parameter is 5, which means each charge spans 5 grid cells in each dimension. The larger the value of this parameter, the smaller the FFT grid will need to be to achieve the requested precision. Conversely, the smaller the order value, the larger the grid will be. Note that there is an inherent trade−off involved: a small grid will lower the cost of FFTs, but a large order parameter will increase the cost of intepolating charge/fields to/from the grid. And vice versa.

The order parameter may be reset by LAMMPS when it sets up the PPPM FFT grid if the implied grid stencil extends beyond the grid cells owned by neighboring processors. Typically this will only occur when small problems are run on large numbers of processors. A warning will be generated indicating the order parameter is being reduced to allow LAMMPS to run the problem.

The *gewald* keyword sets the value of the Ewald or PPPM G−ewald parameter. Without this setting, LAMMPS chooses the parameter automatically as a function of cutoff, precision, grid spacing, etc. This means it can vary from one simulation to the next which may not be desirable for matching a KSpace solver to a pre−tabulated pairwise potential. This setting can also be useful if Ewald or PPPM fails to choose a good grid spacing and G−ewald parameter automatically. If the value is set to 0.0, LAMMPS will choose the G−ewald parameter automatically.

The *slab* keyword allows an Ewald or PPPM solver to be used for a systems that are periodic in x,y but non−periodic in z − a boundary setting of "boundary p p f". This is done by treating the system as if it were periodic in z, but inserting empty volume between atom slabs and removing dipole inter−slab interactions so that slab−slab interactions are effectively turned off. The volfactor value sets the ratio of the extended dimension in z divided by the actual dimension in z. The recommended value is 3.0. A larger value is inefficient; a smaller value introduces unwanted slab−slab interactions. The use of fixed boundaries in z means that the user must prevent particle migration beyond the initial z−bounds, typically by providing a wall−style fix. The methodology behind the *slab* option is explained in the paper by (Yeh).

Restrictions: none

Related commands:

kspace_style, boundary

Default:

The option defaults are mesh = 0 0 0, order = 5, gewald = 0.0, and slab = 1.0.

(Yeh) Yeh and Berkowitz, J Chem Phys, 111, 3155 (1999).

kspace_style command

Syntax:

```
kspace_style style value
```

- style = *none* or *ewald* or *pppm* or *pppm/cg* or *pppm/tip4p* or *ewald/n* or *pppm/gpu*

```
  none value = none
    ewald value = precision
      precision = desired accuracy
    pppm value = precision
      precision = desired accuracy
    pppm/cg value = precision (smallq)
      precision = desired accuracy
      smallq = cutoff for charges to be considered (optional) (charge units)
    pppm/tip4p value = precision
      precision = desired accuracy
    ewald/n value = precision
      precision = desired accuracy
    pppm/gpu value = precision
      precision = desired accuracy
```

Examples:

```
kspace_style pppm 1.0e-4
kspace_style pppm/cg 1.0e-5 1.0e-6
kspace_style none
```

Description:

Define a K−space solver for LAMMPS to use each timestep to compute long−range Coulombic interactions or long−range 1/r^N interactions. When such a solver is used in conjunction with an appropriate pair style, the cutoff for Coulombic or other 1/r^N interactions is effectively infinite; each charge in the system interacts with charges in an infinite array of periodic images of the simulation domain.

The *ewald* style performs a standard Ewald summation as described in any solid−state physics text.

The *pppm* style invokes a particle−particle particle−mesh solver (Hockney) which maps atom charge to a 3d mesh, uses 3d FFTs to solve Poisson's equation on the mesh, then interpolates electric fields on the mesh points back to the atoms. It is closely related to the particle−mesh Ewald technique (PME) (Darden) used in AMBER and CHARMM. The cost of traditional Ewald summation scales as N^(3/2) where N is the number of atoms in the system. The PPPM solver scales as Nlog(N) due to the FFTs, so it is almost always a faster choice (Pollock).

The *pppm/cg* style is identical to the *pppm* style except that it has an optimization for systems where most particles are uncharged. The optional *smallq* argument defines the cutoff for the absolute charge value which determines whether a particle is considered charged or not. Its default value is 1.0e−5.

The *pppm/tip4p* style is identical to the *pppm* style except that it adds a charge at the massless 4th site in each TIP4P water molecule. It should be used with pair styles with a *long/tip4p* in their style name.

The *ewald/n* style augments *ewald* by adding long−range dispersion sum capabilities for 1/r^N potentials and is useful for simulation of interfaces (Veld). It also performs standard coulombic Ewald summations, but in a more

efficient manner than the *ewald* style. The 1/r^N capability means that Lennard−Jones or Buckingham potentials can be used with *ewald/n* without a cutoff, i.e. they become full long−range potentials.

Currently, only the *ewald/n* style can be used with non−orthogonal (triclinic symmetry) simulation boxes.

Note that the PPPM styles can be used with single−precision FFTs by using the compiler switch −DFFT_SINGLE for the FFT_INC setting in your lo−level Makefile. This setting also changes some of the PPPM operations (e.g. mapping charge to mesh and interpolating electric fields to particles) to be performed in single precision. This option can speed−up long−range calulations, particularly in parallel or on GPUs. The use of the −DFFT_SINGLE flag is discussed in this section of the manual.

When a kspace style is used, a pair style that includes the short−range correction to the pairwise Coulombic or other 1/r^N forces must also be selected. For Coulombic interactions, these styles are ones that have a *coul/long* in their style name. For 1/r^6 dispersion forces in a Lennard−Jones or Buckingham potential, see the pair_style lj/coul or pair_style buck/coul commands.

A precision value of 1.0e−4 means one part in 10000. This setting is used in conjunction with the pairwise cutoff to determine the number of K−space vectors for style *ewald* or the FFT grid size for style *pppm*.

See the kspace_modify command for additional options of the K−space solvers that can be set.

Styles with a *cuda*, *gpu*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in this section of the manual. The accelerated styles take the same arguments and should produce the same results, except for round−off and precision issues.

More specifically, the *pppm/gpu* style performs charge assignment and force interpolation calculations on the GPU. These processes are performed either in single or double precision, depending on whether the −DFFT_SINGLE setting was specified in your lo−level Makefile, as discussed above. The FFTs themselves are still calculated on the CPU. If *pppm/gpu* is used with a GPU−enabled pair style, part of the PPPM calculation can be performed concurrently on the GPU while other calculations for non−bonded and bonded force calculation are performed on the CPU.

These accelerated styles are part of the "user−cuda", "gpu", and "opt" packages respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

See this section of the manual for more instructions on how to use the accelerated styles effectively.

Restrictions:

A simulation must be 3d and periodic in all dimensions to use an Ewald or PPPM solver. The only exception is if the slab option is set with kspace_modify, in which case the xy dimensions must be periodic and the z dimension must be non−periodic.

Kspace styles are part of the "kspace" package. They are only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

The *ewald/n* style is part of the "user−ewaldn" package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

When using a long−range pairwise TIP4P potential, you must use kspace style *pppm/tip4p* and vice versa.

Related commands:

kspace_modify, pair_style lj/cut/coul/long, pair_style lj/charmm/coul/long, pair_style lj/coul, pair_style buck/coul/long

Default:

```
kspace_style none
```

(Darden) Darden, York, Pedersen, J Chem Phys, 98, 10089 (1993).

(Hockney) Hockney and Eastwood, Computer Simulation Using Particles, Adam Hilger, NY (1989).

(Pollock) Pollock and Glosli, Comp Phys Comm, 95, 93 (1996).

(Veld) In 't Veld, Ismail, Grest, J Chem Phys, in press (2007).

label command

Syntax:

```
label ID
```

- ID = string used as label name

Examples:

```
label xyz
label loop
```

Description:

Label this line of the input script with the chosen ID. Unless a jump command was used previously, this does nothing. But if a jump command was used with a label argument to begin invoking this script file, then all command lines in the script prior to this line will be ignored. I.e. execution of the script will begin at this line. This is useful for looping over a section of the input script as discussed in the jump command.

Restrictions: none

Related commands: none

Default: none

lattice command

Syntax:

```
lattice style scale keyword values ...
```

- style = *none* or *sc* or *bcc* or *fcc* or *hp* or *diamond* or *sq* or *sq2* or *hex* or *custom*
- scale = scale factor between lattice and simulation box

```
for style none:
  scale is not specified (nor any optional arguments)
for all other styles:
  scale = reduced density rho* (for LJ units)
  scale = lattice constant in distance units (for non−LJ units)
```

- zero or more keyword/value pairs may be appended
- keyword = *origin* or *orient* or *spacing* or *a1* or *a2* or *a3* or *basis*

```
origin values = x y z
    x,y,z = fractions of a unit cell (0 <= x,y,z <1)
  orient values = dim i j k
    dim = x or y or z
    i,j,k = integer lattice directions
  spacing values = dx dy dz
    dx,dy,dz = lattice spacings in the x,y,z box directions
  a1,a2,a3 values = x y z
    x,y,z = primitive vector components that define unit cell
  basis values = x y z
    x,y,z = fractional coords of a basis atom (0  <= x,y,z <1)
```

Examples:

```
lattice fcc 3.52
lattice hex 0.85
lattice sq 0.8 origin 0.0 0.5 0.0 orient x 1 1 0 orient y −1 1 0
lattice custom 3.52 a1 1.0 0.0 0.0 a2 0.5 1.0 0.0 a3 0.0 0.0 0.5 &                    basis 0.0 0.0
lattice none
```

Description:

Define a lattice for use by other commands. In LAMMPS, a lattice is simply a set of points in space, determined by a unit cell with basis atoms, that is replicated infinitely in all dimensions. The arguments of the lattice command can be used to define a wide variety of crystallographic lattices.

A lattice is used by LAMMPS in two ways. First, the create_atoms command creates atoms on the lattice points inside the simulation box. Note that the create_atoms command allows different atom types to be assigned to different basis atoms of the lattice. Second, the lattice spacing in the x,y,z dimensions implied by the lattice, can be used by other commands as distance units (e.g. create_box, region and velocity), which are often convenient to use when the underlying problem geometry is atoms on a lattice.

The lattice style must be consistent with the dimension of the simulation − see the dimension command. Styles *sc* or *bcc* or *fcc* or *hcp* or *diamond* are for 3d problems. Styles *sq* or *sq2* or *hex* are for 2d problems. Style *custom* can be used for either 2d or 3d problems.

A lattice consists of a unit cell, a set of basis atoms within that cell, and a set of transformation parameters (scale, origin, orient) that map the unit cell into the simulation box. The vectors a1,a2,a3 are the edge vectors of the unit cell. This is the nomenclature for "primitive" vectors in solid−state crystallography, but in LAMMPS the unit cell they determine does not have to be a "primitive cell" of minimum volume.

Lattices of style *sc*, *fcc*, *bcc*, and *diamond* are 3d lattices that define a cubic unit cell with edge length = 1.0. This means a1 = 1 0 0, a2 = 0 1 0, and a3 = 0 0 1. Style *hcp* has a1 = 1 0 0, a2 = 0 sqrt(3) 0, and a3 = 0 0 sqrt(8/3). The placement of the basis atoms within the unit cell are described in any solid−state physics text. A *sc* lattice has 1 basis atom at the lower−left−bottom corner of the cube. A *bcc* lattice has 2 basis atoms, one at the corner and one at the center of the cube. A *fcc* lattice has 4 basis atoms, one at the corner and 3 at the cube face centers. A *hcp* lattice has 4 basis atoms, two in the z = 0 plane and 2 in the z = 0.5 plane. A *diamond* lattice has 8 basis atoms.

Lattices of style *sq* and *sq2* are 2d lattices that define a square unit cell with edge length = 1.0. This means a1 = 1 0 0 and a2 = 0 1 0. A *sq* lattice has 1 basis atom at the lower−left corner of the square. A *sq2* lattice has 2 basis atoms, one at the corner and one at the center of the square. A *hex* style is also a 2d lattice, but the unit cell is rectangular, with a1 = 1 0 0 and a2 = 0 sqrt(3) 0. It has 2 basis atoms, one at the corner and one at the center of the rectangle.

A lattice of style *custom* allows you to specify a1, a2, a3, and a list of basis atoms to put in the unit cell. By default, a1 and a2 and a3 are 3 orthogonal unit vectors (edges of a unit cube). But you can specify them to be of any length and non−orthogonal to each other, so that they describe a tilted parallelepiped. Via the *basis* keyword you add atoms, one at a time, to the unit cell. Its arguments are fractional coordinates (0.0 <= x,y,z < 1.0), so that a value of 0.5 means a position half−way across the unit cell in that dimension.

This sub−section discusses the arguments that determine how the idealized unit cell is transformed into a lattice of points within the simulation box.

The *scale* argument determines how the size of the unit cell will be scaled when mapping it into the simulation box. I.e. it determines a multiplicative factor to apply to the unit cell, to convert it to a lattice of the desired size and distance units in the simulation box. The meaning of the *scale* argument depends on the units being used in your simulation.

For all unit styles except *lj*, the scale argument is specified in the distance units defined by the unit style. For example, in *real* or *metal* units, if the unit cell is a unit cube with edge length 1.0, specifying scale = 3.52 would create a cubic lattice with a spacing of 3.52 Angstroms. In *cgs* units, the spacing would be 3.52 cm.

For unit style *lj*, the scale argument is the Lennard−Jones reduced density, typically written as rho*. LAMMPS converts this value into the multiplicative factor via the formula "factor^dim = rho/rho*", where rho = N/V with V = the volume of the lattice unit cell and N = the number of basis atoms in the unit cell (described below), and dim = 2 or 3 for the dimensionality of the simulation. Effectively, this means that if LJ particles of size sigma = 1.0 are used in the simulation, the lattice of particles will be at the desired reduced density.

The *origin* option specifies how the unit cell will be shifted or translated when mapping it into the simulation box. The x,y,z values are fractional values (0.0 <= x,y,z < 1.0) meaning shift the lattice by a fraction of the lattice spacing in each dimension. The meaning of "lattice spacing" is discussed below.

The *orient* option specifies how the unit cell will be rotated when mapping it into the simulation box. The *dim* argument is one of the 3 coordinate axes in the simulation box. The other 3 arguments are the crystallographic direction in the lattice that you want to orient along that axis, specified as integers. E.g. "orient x 2 1 0" means the x−axis in the simulation box will be the [210] lattice direction. The 3 lattice directions you specify must be mutually orthogonal and obey the right−hand rule, i.e. (X cross Y) points in the Z direction. Note that this description is really only valid for orthogonal lattices. If you are using the more general lattice style *custom* with

non−orthogonal a1,a2,a3 vectors, then think of the 3 *orient* options as creating a 3x3 rotation matrix which is applied to a1,a2,a3 to rotate the original unit cell to a new orientation in the simulation box.

Several LAMMPS commands have the option to use distance units that are inferred from "lattice spacing" in the x,y,z box directions. E.g. the region command can create a block of size 10x20x20, where 10 means 10 lattice spacings in the x direction.

The *spacing* option sets the 3 lattice spacings directly. All must be non−zero (use 1.0 for dz in a 2d simulation). The specified values are multiplied by the multiplicative factor described above that is associated with the scale factor. Thus a spacing of 1.0 means one unit cell independent of the scale factor. This option can be useful if the spacings LAMMPS computes are inconvenient to use in subsequent commands, which can be the case for non−orthogonal or rotated lattices.

If the *spacing* option is not specified, the lattice spacings are computed by LAMMPS in the following way. A unit cell of the lattice is mapped into the simulation box (scaled, shifted, rotated), so that it now has (perhaps) a modified size and orientation. The lattice spacing in X is defined as the difference between the min/max extent of the x coordinates of the 8 corner points of the modified unit cell. Similarly, the Y and Z lattice spacings are defined as the difference in the min/max of the y and z coordinates.

Note that if the unit cell is orthogonal with axis−aligned edges (not rotated via the *orient* keyword), then the lattice spacings in each dimension are simply the scale factor (described above) multiplied by the length of a1,a2,a3. Thus a *hex* style lattice with a scale factor of 3.0 Angstroms, would have a lattice spacing of 3.0 in x and 3*sqrt(3.0) in y.

IMPORTANT NOTE: For non−orthogonal unit cells and/or when a rotation is applied via the *orient* keyword, then the lattice spacings may be less intuitive. In particular, in these cases, there is no guarantee that the lattice spacing is an integer multiple of the periodicity of the lattice in that direction. Thus, if you create an orthogonal periodic simulation box whose size in a dimension is a multiple of the lattice spacing, and then fill it with atoms via the create_atoms command, you will NOT necessarily create a periodic system. I.e. atoms may overlap incorrectly at the faces of the simulation box.

Regardless of these issues, the values of the lattice spacings LAMMPS calculates are printed out, so their effect in commands that use the spacings should be decipherable.

The command "lattice none" can be used to turn off a previous lattice definition. Any command that attempts to use the lattice directly (create_atoms) or associated lattice spacings will then generate an error. No additional arguments need be used with "lattice none".

Restrictions:

The *a1,a2,a3,basis* keywords can only be used with style *custom*.

Related commands:

dimension, create_atoms, region

Default:

```
lattice none
```

For other lattice styles, the option defaults are origin = 0.0 0.0 0.0, orient = x 1 0 0, orient = y 0 1 0, orient = z 0 0 1, a1 = 1 0 0, a2 = 0 1 0, and a3 = 0 0 1.

log command

Syntax:

```
log file
```

- file = name of new logfile

Examples:

```
log log.equil
```

Description:

This command closes the current LAMMPS log file, opens a new file with the specified name, and begins logging information to it. If the specified file name is *none*, then no new log file is opened.

If multiple processor partitions are being used, the file name should be a variable, so that different processors do not attempt to write to the same log file.

The file "log.lammps" is the default log file for a LAMMPS run. The name of the initial log file can also be set by the command−line switch −log. See this section for details.

Restrictions: none

Related commands: none

Default:

The default LAMMPS log file is named log.lammps

mass command

Syntax:

```
mass I value
```

- I = atom type (see asterisk form below)
- value = mass

Examples:

```
mass 1 1.0
mass * 62.5
mass 2* 62.5
```

Description:

Set the mass for all atoms of one or more atom types. Per−type mass values can also be set in the read_data data file using the "Masses" keyword. See the units command for what mass units to use.

The I index can be specified in one of two ways. An explicit numeric value can be used, as in the 1st example above. Or a wild−card asterisk can be used to set the mass for multiple atom types. This takes the form "*" or "*n" or "n*" or "m*n". If N = the number of atom types, then an asterisk with no numeric values means all types from 1 to N. A leading asterisk means all types from 1 to n (inclusive). A trailing asterisk means all types from n to N (inclusive). A middle asterisk means all types from m to n (inclusive).

A line in a data file that follows the "Masses" keyword specifies mass using the same format as the arguments of the mass command in an input script, except that no wild−card asterisk can be used. For example, under the "Masses" section of a data file, the line that corresponds to the 1st example above would be listed as

```
1 1.0
```

Note that the mass command can only be used if the atom style requires per−type atom mass to be set. Currently, all but the *sphere* and *ellipsoid* and *peri* styles do. They require mass to be set for individual particles, not types. Per−atom masses are defined in the data file read by the read_data command, or set to default values by the create_atoms command. Per−atom masses can also be set to new values by the set mass or set density commands.

Also note that pair_style eam defines the masses of atom types in the EAM potential file, in which case the mass command is normally not used.

If you define a hybrid atom style which includes one (or more) sub−styles which require per−type mass and one (or more) sub−styles which require per−atom mass, then you must define both. However, in this case the per−type mass will be ignored; only the per−atom mass will be used by LAMMPS.

Restrictions:

This command must come after the simulation box is defined by a read_data, read_restart, or create_box command.

All masses must be defined before a simulation is run. They must also all be defined before a velocity or fix shake command is used.

The mass assigned to any type or atom must be > 0.0.

Related commands: none

Default: none

min_modify command

Syntax:

```
min_modify keyword values ...
```

- one or more keyword/value pairs may be listed

```
keyword = dmax or line
  dmax value = max
    max = maximum distance for line search to move (distance units)
  line value = backtrack or quadratic
    backtrack,quadratic = style of linesearch to use
```

Examples:

```
min_modify dmax 0.2
```

Description:

This command sets parameters that affect the energy minimization algorithms selected by the min_style command. The various settings may affect the convergence rate and overall number of force evaluations required by a minimization, so users can experiment with these parameters to tune their minimizations.

The *cg* and *sd* minimization styles have an outer iteration and an inner iteration which is steps along a one−dimensional line search in a particular search direction. The *dmax* parameter is how far any atom can move in a single line search in any dimension (x, y, or z). For the *quickmin* and *fire* minimization styles, the *dmax* setting is how far any atom can move in a single iteration (timestep). Thus a value of 0.1 in real units means no atom will move further than 0.1 Angstroms in a single outer iteration. This prevents highly overlapped atoms from being moved long distances (e.g. through another atom) due to large forces.

The choice of line search algorithm for the *cg* and *sd* minimization styles can be selected via the *line* keyword. The default backtracking search is robust and should always find a local energy minimum. However, it will "converge" when it can no longer reduce the energy of the system. Individual atom forces may still be larger than desired at this point, because the energy change is measured as the difference of two large values (energy before and energy after) and that difference may be smaller than machine epsilon even if atoms could move in the gradient direction to reduce forces further.

By contast, the *quadratic* line search algorithm is often able to reduce forces closer to 0.0. It may also be more efficient than the backtracking algorithm by requiring fewer energy/force evaluations. However, it may not be as robust for some problems.

Restrictions: none

Related commands:

min_style, minimize

Default:

The option defaults are dmax = 0.1 and line = backtrack.

min_style command

Syntax:

```
min_style style
```

- style = *cg* or *hftn* or *sd* or *quickmin* or *fire*

Examples:

```
min_style cg
min_style fire
```

Description:

Choose a minimization algorithm to use when a minimize command is performed.

Style *cg* is the Polak−Ribiere version of the conjugate gradient (CG) algorithm. At each iteration the force gradient is combined with the previous iteration information to compute a new search direction perpendicular (conjugate) to the previous search direction. The PR variant affects how the direction is chosen and how the CG method is restarted when it ceases to make progress. The PR variant is thought to be the most effective CG choice for most problems.

Style *hftn* is a Hessian−free truncated Newton algorithm. At each iteration a quadratic model of the energy potential is solved by a conjugate gradient inner iteration. The Hessian (second derivatives) of the energy is not formed directly, but approximated in each conjugate search direction by a finite difference directional derivative. When close to an energy minimum, the algorithm behaves like a Newton method and exhibits a quadratic convergence rate to high accuracy. In most cases the behavior of *hftn* is similar to *cg*, but it offers an alternative if *cg* seems to perform poorly. This style is not affected by the min_modify command.

Style *sd* is a steepest descent algorithm. At each iteration, the search direction is set to the downhill direction corresponding to the force vector (negative gradient of energy). Typically, steepest descent will not converge as quickly as CG, but may be more robust in some situations.

Style *quickmin* is a damped dynamics method described in (Sheppard), where the damping parameter is related to the projection of the velocity vector along the current force vector for each atom. The velocity of each atom is initialized to 0.0 by this style, at the beginning of a minimization.

Style *fire* is a damped dynamics method described in (Bitzek), which is similar to *quickmin* but adds a variable timestep and alters the projection operation to maintain components of the velocity non−parallel to the current force vector. The velocity of each atom is initialized to 0.0 by this style, at the beginning of a minimization.

Either the *quickmin* and *fire* styles are useful in the context of nudged elastic band (NEB) calculations via the neb command.

IMPORTANT NOTE: The *quickmin* and *fire* styles do not yet support the use of the fix box/relax command or minimizations involving the electron radius in eFF models.

Restrictions: none

Related commands:

min_modify, minimize, neb

Default:

```
min_style cg
```

(Sheppard) Sheppard, Terrell, Henkelman, J Chem Phys, 128, 134106 (2008). See ref 1 in this paper for original reference to Qmin in Jonsson, Mills, Jacobsen.

(Bitzek) Bitzek, Koskinen, Gahler, Moseler, Gumbsch, Phys Rev Lett, 97, 170201 (2006).

minimize command

Syntax:

```
minimize etol ftol maxiter maxeval
```

- etol = stopping tolerance for energy (unitless)
- ftol = stopping tolerance for force (force units)
- maxiter = max iterations of minimizer
- maxeval = max number of force/energy evaluations

Examples:

```
minimize 1.0e-4 1.0e-6 100 1000
minimize 0.0 1.0e-8 1000 100000
```

Description:

Perform an energy minimization of the system, by iteratively adjusting atom coordinates. Iterations are terminated when one of the stopping criteria is satisfied. At that point the configuration will hopefully be in local potential energy minimum. More precisely, the configuration should approximate a critical point for the objective function (see below), which may or may not be a local minimum.

The minimization algorithm used is set by the min_style command. Other options are set by the min_modify command. Minimize commands can be interspersed with run commands to alternate between relaxation and dynamics. The minimizers bound the distance atoms move in one iteration, so that you can relax systems with highly overlapped atoms (large energies and forces) by pushing the atoms off of each other.

Alternate means of relaxing a system are to run dynamics with a small or limited timestep. Or dynamics can be run using fix viscous to impose a damping force that slowly drains all kinetic energy from the system. The pair_style soft potential can be used to un−overlap atoms while running dynamics.

The minimization styles *cg*, *sd*, and *hftn* involves an outer iteration loop which sets the search direction along which atom coordinates are changed. An inner iteration is then performed using a line search algorithm. The line search typically evaluates forces and energies several times to set new coordinates. Currently, a backtracking algorithm is used which may not be optimal in terms of the number of force evaulations performed, but appears to be more robust than previous line searches we've tried. The backtracking method is described in Nocedal and Wright's Numerical Optimization (Procedure 3.1 on p 41).

The minimization styles *quickmin* and *fire* perform damped dynamics using an Euler integration step. Thus they require a timestep be defined, typically the same value used for running dynamics with the system.

The objective function being minimized is the total potential energy of the system as a function of the N atom coordinates:

$$E(r_1, r_2, \ldots, r_N) = \sum_{i,j} E_{pair}(r_i, r_j) + \sum_{ij} E_{bond}(r_i, r_j) + \sum_{ijk} E_{angle}(r_i, r_j, r_k) + \sum_{ijkl} E_{dihedral}(r_i, r_j, r_k, r_l) + \sum_{ijkl} E_{improper}(r_i, r_j, r_k, r_l) + \sum_{i} E_{fix}(r_i)$$

where the first term is the sum of all non−bonded pairwise interactions including long−range Coulombic interactions, the 2nd thru 5th terms are bond, angle, dihedral, and improper interactions respectively, and the last term is energy due to fixes which can act as constraints or apply force to atoms, such as thru interaction with a wall. See the discussion below about how fix commands affect minimization.

The starting point for the minimization is the current configuration of the atoms.

The minimization procedure stops if any of several criteria are met:

- the change in energy between outer iterations is less than *etol*
- the 2−norm (length) of the global force vector is less than the *ftol*
- the line search fails because the step distance backtracks to 0.0
- the number of outer iterations or timesteps exceeds *maxiter*
- the number of total force evaluations exceeds *maxeval*

For the first criterion, the specified energy tolerance *etol* is unitless; it is met when the energy change between successive iterations divided by the energy magnitude is less than or equal to the tolerance. For example, a setting of 1.0e−4 for *etol* means an energy tolerance of one part in 10^4. For the damped dynamics minimizers this check is not performed for a few steps after velocities are reset to 0, otherwise the minimizer would prematurely converge.

For the second criterion, the specified force tolerance *ftol* is in force units, since it is the length of the global force vector for all atoms, e.g. a vector of size 3N for N atoms. Since many of the components will be near zero after minimization, you can think of *ftol* as an upper bound on the final force on any component of any atom. For example, a setting of 1.0e−4 for *ftol* means no x, y, or z component of force on any atom will be larger than 1.0e−4 (in force units) after minimization.

Either or both of the *etol* and *ftol* values can be set to 0.0, in which case some other criterion will terminate the minimization.

During a minimization, the outer iteration count is treated as a timestep. Output is triggered by this timestep, e.g. thermodynamic output or dump and restart files.

Using the thermo_style custom command with the *fmax* or *fnorm* keywords can be useful for monitoring the progress of the minimization. Note that these outputs will be calculated only from forces on the atoms, and will not include any extra degrees of freedom, such as from the fix box/relax command.

Following minimization, a statistical summary is printed that lists which convergence criterion caused the minimizer to stop, as well as information about the energy, force, final line search, and and iteration counts. An example is as follows:

```
Minimization stats:
  Stopping criterion = max iterations
  Energy initial, next-to-last, final =
       -0.626828169302     -2.82642039062     -2.82643549739
  Force two-norm initial, final = 2052.1 91.9642
  Force max component initial, final = 346.048 9.78056
  Final line search alpha, max atom move = 2.23899e-06 2.18986e-05
  Iterations, force evaluations = 2000 12724
```

The 3 energy values are for before and after the minimization and on the next−to−last iteration. This is what the *etol* parameter checks.

The two−norm force values are the length of the global force vector before and after minimization. This is what the *ftol* parameter checks.

The max−component force values are the absolute value of the largest component (x,y,z) in the global force vector, i.e. the infinity−norm of the force vector.

The alpha parameter for the line−search, when multiplied by the max force component (on the last iteration), gives the max distance any atom moved during the last iteration. Alpha will be 0.0 if the line search could not reduce the energy. Even if alpha is non−zero, if the "max atom move" distance is tiny compared to typical atom coordinates, then it is possible the last iteration effectively caused no atom movement and thus the evaluated energy did not change and the minimizer terminated. Said another way, even with non−zero forces, it's possible the effect of those forces is to move atoms a distance less than machine precision, so that the energy cannot be further reduced.

The iterations and force evaluation values are what is checked by the *maxiter* and *maxeval* parameters.

IMPORTANT NOTE: There are several force fields in LAMMPS which have discontinuities or other approximations which may prevent you from performing an energy minimization to high tolerances. For example, you should use a pair style that goes to 0.0 at the cutoff distance when performing minimization (even if you later change it when running dynamics). If you do not do this, the total energy of the system will have discontinuities when the relative distance between any pair of atoms changes from cutoff+epsilon to cutoff−epsilon and the minimizer may behave poorly. Some of the manybody potentials use splines and other internal cutoffs that inherently have this problem. The long−range Coulombic styles (PPPM, Ewald) are approximate to within the user−specified tolerance, which means their energy and forces may not agree to a higher precision than the Kspace−specified tolerance. In all these cases, the minimizer may give up and stop before finding a minimum to the specified energy or force tolerance.

Note that a cutoff Lennard−Jones potential (and others) can be shifted so that its energy is 0.0 at the cutoff via the pair_modify command. See the doc pages for inidividual pair styles for details. Note that Coulombic potentials always have a cutoff, unless versions with a long−range component are used (e.g. pair_style lj/cut/coul/long). The CHARMM potentials go to 0.0 at the cutoff (e.g. pair_style lj/charmm/coul/charmm), as do the GROMACS potentials (e.g. pair_style lj/gromacs).

If a soft potential (pair_style soft) is used the Astop value is used for the prefactor (no time dependence).

The fix box/relax command can be used to apply an external pressure to the simulation box and allow it to shrink/expand during the minimization.

Only a few other fixes (typically those that apply force constraints) are invoked during minimization. See the doc pages for individual fix commands to see which ones are relevant.

IMPORTANT NOTE: Some fixes which are invoked during minimization have an associated potential energy. For that energy to be included in the total potential energy of the system (the quantity being minimized), you MUST enable the fix_modify *energy* option for that fix. The doc pages for individual fix commands specify if this should be done.

Restrictions:

Features that are not yet implemented are listed here, in case someone knows how they could be coded:

It is an error to use fix shake with minimization because it turns off bonds that should be included in the potential energy of the system. The effect of a fix shake can be approximated during a minimization by using stiff spring

constants for the bonds and/or angles that would normally be constrained by the SHAKE algorithm.

Fix rigid is also not supported by minimization. It is not an error to have it defined, but the energy minimization will not keep the defined body(s) rigid during the minimization. Note that if bonds, angles, etc internal to a rigid body have been turned off (e.g. via neigh_modify exclude), they will not contribute to the potential energy which is probably not what is desired.

Pair potentials that produce torque on a particle (e.g. granular potentials or the GayBerne potential for ellipsoidal particles) are not relaxed by a minimization. More specifically, radial relaxations are induced, but no rotations are induced by a minimization, so such a system will not fully relax.

Related commands:

min_modify, min_style, run_style

Default: none

neb command

Syntax:

```
neb etol ftol N1 N2 Nevery filename
```

- etol = stopping tolerance for energy (energy units)
- ftol = stopping tolerance for force (force units)
- N1 = max # of iterations (timesteps) to run initial NEB
- N2 = max # of iterations (timesteps) to run barrier−climbing NEB
- Nevery = print replica energies and reaction coordinates every this many timesteps
- filename = file specifying final atom coordinates on other side of barrier

Examples:

```
neb 0.1 0.0 1000 500 50 coords.final
neb 0.0 0.001 1000 500 50 coords.final
```

Description:

Perform a nudged elastic band (NEB) calculation using multiple replicas of a system. Two or more replicas must be used, two of which are the end points of the transition path.

NEB is a method for finding both the atomic configurations and height of the energy barrier associated with a transition state, e.g. for an atom to perform a diffusive hop from one energy basin to another in a coordinated fashion with its neighbors. The implementation in LAMMPS follows the discussion in these 3 papers: (Henkelman1), (Henkelman2), and (Nakano).

Each replica runs on a partition of one or more processors. Processor partitions are defined at run−time using the −partition command−line switch; see this section of the manual. Note that if you have MPI installed, you can run a multi−replica simulation with more replicas (partitions) than you have physical processors, e.g you can run a 10−replica simulation on one or two processors. You will simply not get the performance speed−up you would see with one or more physical processors per replica. See this section of the manual for further discussion.

NOTE: The current NEB implementation in LAMMPS restricts you to having exactly one processor per replica.

When a NEB calculation is performed, it is assumed that each replica is running the same model, though LAMMPS does not check for this. I.e. the simulation domain, the number of atoms, the interaction potentials, and the starting configuration when the neb command is issued should be the same for every replica.

In a NEB calculation each atom in a replica is connected to the same atom in adjacent replicas by springs, which induce inter−replica forces. These forces are imposed by the fix neb command, which must be used in conjunction with the neb command. The group used to define the fix neb command specifies which atoms the inter−replica springs are applied to. These are the NEB atoms. Additional atoms can be present in your system, e.g. to provide a background force field or simply to hold fixed during the NEB procedure, but they will not be part of the barrier finding procedure.

The "starting configuration" for NEB should be a state with the NEB atoms (and all other atoms) having coordinates on one side of the energy barrier. These coordinates will be assigned to the first replica #1. The coordinates should be close to a local energy minimum. A perfect energy minimum is not required, since NEB

runs via damped dynamics which will tend to drive the configuration of replica #1 to a true energy minimum, but you will typically get better convergence if the initial state is already at a minimum. For example, for a system with a free surface, the surface should be fully relaxed before attempting a NEB calculation.

The final configuration is specified in the input *filename*, which is formatted as described below. Only coordinates for NEB atoms or a subset of them should be listed in the file; they represent the state of the system on the other side of the barrier, at or near an energy minimum. These coordinates will be assigned to the last replica #M. The final coordinates of atoms not listed in *filename* are set equal to their initial coordinates. Again, a perfect energy minimum is not required for the final configuration, since the atoms in replica #M will tend to move during the NEB procedure to the nearest energy minimum. Also note that a final coordinate does not need to be specified for a NEB atom if you expect it to only displace slightly during the NEB procedure. For example, only the final coordinate of the single atom diffusing into a vacancy need be specified if the surrounding atoms will only relax slightly in the final configuration.

The initial coordinates of all atoms (not just NEB atoms) in the intermediate replicas #2,#3,...,#M−1 are set to values linearly interpolated between the corresponding atoms in replicas #1 and #M.

A NEB calculation has two stages, each of which is a minimization procedure, performed via damped dynamics. To enable this, you must first define an appropriate min_style, such as *quickmin* or *fire*. The *cg*, *sd*, and *hftn* styles cannot be used, since they perform iterative line searches in their inner loop, which cannot be easily synchronized across multiple replicas.

The minimizer tolerances for energy and force are set by *etol* and *ftol*, the same as for the minimize command.

A non−zero *etol* means that the NEB calculation will terminate if the energy criterion is met by every replica. The energies being compared to *etol* do not include any contribution from the inter−replica forces, since these are non−conservative. A non−zero *ftol* means that the NEB calculation will terminate if the force criterion is met by every replica. The forces being compared to *ftol* include the inter−replica forces between an atom and its images in adjacent replicas.

The maximum number of iterations in each stage is set by *N1* and *N2*. These are effectively timestep counts since each iteration of damped dynamics is like a single timestep in a dynamics run. During both stages, the potential energy of each replica and its normalized distance along the reaction path (reaction coordinate RD) will be printed to the screen and log file every *Nevery* timesteps. The RD is 0 and 1 for the first and last replica. For intermediate replicas, it is the cumulative distance (normalized by the total cumulative distance) between adjacent replicas, where "distance" is defined as the length of the 3N−vector of differences in atomic coordinates, where N is the number of NEB atoms involved in the transition. These outputs allow you to monitor NEB's progress in finding a good energy barrier. *N1* and *N2* must both be multiples of *Nevery*.

In the first stage of NEB, the set of replicas should converge toward the minimum energy path (MEP) of conformational states that transition over the barrier. The MEP for a barrier is defined as a sequence of 3N−dimensional states that cross the barrier at its saddle point, each of which has a potential energy gradient parallel to the MEP itself. The replica states will also be roughly equally spaced along the MEP due to the inter−replica spring force added by the fix neb command.

In the second stage of NEB, the replica with the highest energy is selected and the inter−replica forces on it are converted to a force that drives its atom coordinates to the top or saddle point of the barrier, via the barrier−climbing calculation described in (Henkelman2). As before, the other replicas rearrange themselves along the MEP so as to be roughly equally spaced.

When both stages are complete, if the NEB calculation was successful, one of the replicas should be an atomic configuration at the top or saddle point of the barrier, the potential energies for the set of replicas should represent

the energy profile of the barrier along the MEP, and the configurations of the replicas should be a sequence of configurations along the MEP.

A few other settings in your input script are required or advised to perform a NEB calculation.

An atom map must be defined which it is not by default for atom_style atomic problems. The atom_modify map command can be used to do this.

The "atom_modify sort 0 0.0" command should be used to turn off atom sorting.

NOTE: This sorting restriction will be removed in a future version of NEB in LAMMPS.

The minimizers in LAMMPS operate on all atoms in your system, even non−NEB atoms, as defined above. To prevent non−NEB atoms from moving during the minimization, you should use the fix setforce command to set the force on each of those atoms to 0.0. This is not required, and may not even be desired in some cases, but if those atoms move too far (e.g. because the initial state of your system was not well−minimized), it can cause problems for the NEB procedure.

The damped dynamics minimizers, such as *quickmin* and *fire*), adjust the position and velocity of the atoms via an Euler integration step. Thus you must define an appropriate timestep to use with NEB. Using the same timestep that would be used for a dynamics run of your system is advised.

The specified *filename* contains atom coordinates for the final configuration. Only atoms with coordinates different than the initial configuration need to be specified, i.e. those geometrically near the barrier.

The file can be ASCII text or a gzipped text file (detected by a .gz suffix). The file should contain one line per atom, formatted with the atom ID, followed by the final x,y,z coordinates:

```
125     24.97311   1.69005     23.46956
126     1.94691    2.79640     1.92799
127     0.15906    3.46099     0.79121
...
```

The lines can be listed in any order.

Four kinds of output can be generated during a NEB calculation: energy barrier statistics, thermodynamic output by each replica, dump files, and restart files.

When running with multiple partitions (each of which is a replica in this case), the print−out to the screen and master log.lammps file contains a line of output, printed once every *Nevery* timesteps. It contains the timestep, the maximum force per replica, the maximum force per atom (in any replica), potential gradients in the initial, final, and climbing replicas, the forward and backward energy barriers, the total reaction coordinate (RDT), and the normalized reaction coordinate and potential energy of each replica.

The "maximum force per replica" is the two−norm of the 3N−length force vector for the atoms in each replica, maximized across replicas, which is what the *ftol* setting is checking against. In this case, N is all the atoms in each replica. The "maximum force per atom" is the maximum force component of any atom in any replica. The potential gradients are the two−norm of the 3N−length force vector solely due to the interaction potential i.e. without adding in inter−replica forces. Note that inter−replica forces are zero in the initial and final replicas, and only affect the direction in the climbing replica. For this reason, the "maximum force per replica" is often equal to the potential gradient in the climbing replica. In the first stage of NEB, there is no climbing replica, and so the potential gradient in the highest energy replica is reported, since this replica will become the climbing replica in the second stage of NEB.

The "reaction coordinate" (RD) for each replica is the two−norm of the 3N−length vector of distances between its atoms and the preceding replica's atoms, added to the RD of the preceding replica. The RD of the first replica RD1 = 0.0; the RD of the final replica RDN = RDT, the total reaction coordinate. The normalized RDs are divided by RDT, so that they form a monotonically increasing sequence from zero to one. When computing RD, N only includes the atoms being operated on by the fix neb command.

The forward (reverse) energy barrier is the potential energy of the highest replica minus the energy of the first (last) replica.

When running on multiple partitions, LAMMPS produces additional log files for each partition, e.g. log.lammps.0, log.lammps.1, etc. For a NEB calculation, these contain the thermodynamic output for each replica.

If dump commands in the input script define a filename that includes a *universe* or *uloop* style variable, then one dump file (per dump command) will be created for each replica. At the end of the NEB calculation, the final snapshot in each file will contain the sequence of snapshots that transition the system over the energy barrier. Earlier snapshots will show the convergence of the replicas to the MEP.

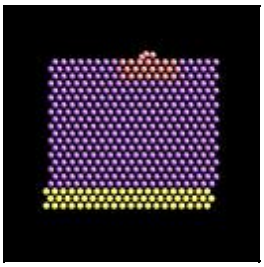
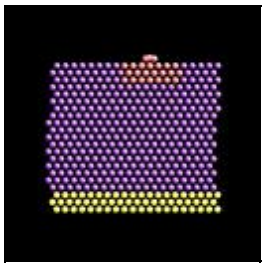
Likewise, restart filenames can be specified with a *universe* or *uloop* style variable, to generate restart files for each replica. These may be useful if the NEB calculation fails to converge properly to the MEP, and you wish to restart the calculation from an intermediate point with altered parameters.

There are 2 Python scripts provided in the tools/python directory, neb_combine.py and neb_final.py, which are useful in analyzing output from a NEB calculation. Assume a NEB simulation with M replicas, and the NEB atoms labelled with a specific atom type.

The neb_combine.py script extracts atom coords for the NEB atoms from all M dump files and creates a single dump file where each snapshot contains the NEB atoms from all the replicas and one copy of non−NEB atoms from the first replica (presumed to be identical in other replicas). This can be visualized/animated to see how the NEB atoms relax as the NEB calculation proceeds.

The neb_final.py script extracts the final snapshot from each of the M dump files to create a single dump file with M snapshots. This can be visualized to watch the system make its transition over the energy barrier.

To illustrate, here are images from the final snapshot produced by the neb_combine.py script run on the dump files produced by the two example input scripts in examples/neb. Click on them to see a larger image.

Restrictions:

This command can only be used if LAMMPS was built with the "replica" package. See the Making LAMMPS section for more info on packages.

Related commands:

prd, temper, fix langevin, fix viscous

Default: none

(Henkelman1) Henkelman and Jonsson, J Chem Phys, 113, 9978−9985 (2000).

(Henkelman2) Henkelman, Uberuaga, Jonsson, J Chem Phys, 113, 9901−9904 (2000).

(Nakano) Nakano, Comp Phys Comm, 178, 280−289 (2008).

neigh_modify command

Syntax:

```
neigh_modify keyword values ...
```

- one or more keyword/value pairs may be listed

```
keyword = delay or every or check or once or include or exclude or page or one or binsize
  delay value = N
    N = delay building until this many steps since last build
  every value = M
    M = build neighbor list every this many steps
  check value = yes or no
    yes = only build if some atom has moved half the skin distance or more
    no = always build on 1st step that every and delay are satisfied
  once
    yes = only build neighbor list once at start of run and never rebuild
    no = rebuild neighbor list according to other settings
  include value = group-ID
    group-ID = only build pair neighbor lists for atoms in this group
  exclude values:
    type M N
      M,N = exclude if one atom in pair is type M, other is type N
    group group1-ID group2-ID
      group1-ID,group2-ID = exclude if one atom is in 1st group, other in 2nd
    molecule group-ID
      groupname = exclude if both atoms are in the same molecule and in the same group
    none
      delete all exclude settings
  page value = N
    N = number of pairs stored in a single neighbor page
  one value = N
    N = max number of neighbors of one atom
  binsize value = size
    size = bin size for neighbor list construction (distance units)
```

Examples:

```
neigh_modify every 2 delay 10 check yes page 100000
neigh_modify exclude type 2 3
neigh_modify exclude group frozen frozen check no
neigh_modify exclude group residue1 chain3
neigh_modify exclude molecule rigid
```

Description:

This command sets parameters that affect the building and use of pairwise neighbor lists.

The *every*, *delay*, *check*, and *once* options affect how often lists are built as a simulation runs. The *delay* setting means never build a new list until at least N steps after the previous build. The *every* setting means build the list every M steps (after the delay has passed). If the *check* setting is *no*, the list is built on the 1st step that satisfies the *delay* and *every* settings. If the *check* setting is *yes*, then the list is only built on a particular step if some atom has moved more than half the skin distance (specified in the neighbor command) since the last build. If the *once* setting is yes, then the neighbor list is only built once at the beginning of each run, and never rebuilt. This should only be done if you are certain atoms will not move far enough that the list should be rebuilt. E.g. running a

simulation of a cold crystal. Note that it is not that expensive to check if neighbor lists should be rebuilt.

When the rRESPA integrator is used (see the run_style command), the *every* and *delay* parameters refer to the longest (outermost) timestep.

The *include* option limits the building of pairwise neighbor lists to atoms in the specified group. This can be useful for models where a large portion of the simulation is particles that do not interact with other particles or with each other via pairwise interactions. The group specified with this option must also be specified via the atom_modify first command.

The *exclude* option turns off pairwise interactions between certain pairs of atoms, by not including them in the neighbor list. These are sample scenarios where this is useful:

- In crack simulations, pairwise interactions can be shut off between 2 slabs of atoms to effectively create a crack.
- When a large collection of atoms is treated as frozen, interactions between those atoms can be turned off to save needless computation. E.g. Using the fix setforce command to freeze a wall or portion of a bio−molecule.
- When one or more rigid bodies are specified, interactions within each body can be turned off to save needless computation. See the fix rigid command for more details.

The *exclude type* option turns off the pairwise interaction if one atom is of type M and the other of type N. M can equal N. The *exclude group* option turns off the interaction if one atom is in the first group and the other is the second. Group1−ID can equal group2−ID. The *exclude molecule* option turns off the interaction if both atoms are in the specified group and in the same molecule, as determined by their molecule ID.

Each of the exclude options can be specified multiple times. The *exclude type* option is the most efficient option to use; it requires only a single check, no matter how many times it has been specified. The other exclude options are more expensive if specified multiple times; they require one check for each time they have been specified.

Note that the exclude options only affect pairwise interactions; see the delete_bonds command for information on turning off bond interactions.

The *page* and *one* options affect how memory is allocated for the neighbor lists. For most simulations the default settings for these options are fine, but if a very large problem is being run or a very long cutoff is being used, these parameters can be tuned. The indices of neighboring atoms are stored in "pages", which are allocated one after another as they fill up. The size of each page is set by the *page* value. A new page is allocated when the next atom's neighbors could potentially overflow the list. This threshold is set by the *one* value which tells LAMMPS the maximum number of neighbor's one atom can have.

IMPORTANT NOTE: LAMMPS can crash without an error message if the number of neighbors for a single particle is larger than the *page* setting, which means it is much, much larger than the *one* setting. This is because LAMMPS doesn't error check these limits for every pairwise interaction (too costly), but only after all the particle's neighbors have been found. This problem usually means something is very wrong with the way you've setup your problem (particle spacing, cutoff length, neighbor skin distance, etc). If you really expect that many neighbors per particle, then boost the *one* and *page* settings accordingly.

The *binsize* option allows you to specify what size of bins will be used in neighbor list construction to sort and find neighboring atoms. By default, for neighbor style bin, LAMMPS uses bins that are 1/2 the size of the maximum pair cutoff. For neighbor style multi, the bins are 1/2 the size of the minimum pair cutoff. Typically these are good values values for minimizing the time for neighbor list construction. This setting overrides the default. If you make it too big, there is little overhead due to looping over bins, but more atoms are checked. If

you make it too small, the optimal number of atoms is checked, but bin overhead goes up. If you set the binsize to 0.0, LAMMPS will use the default binsize of 1/2 the cutoff.

Restrictions:

If the "delay" setting is non−zero, then it must be a multiple of the "every" setting.

The exclude molecule option can only be used with atom styles that define molecule IDs.

The value of the *page* setting must be at least 10x larger than the *one* setting. This insures neighbor pages are not mostly empty space.

Related commands:

neighbor, delete_bonds

Default:

The option defaults are delay = 10, every = 1, check = yes, once = no, include = all, exclude = none, page = 100000, one = 2000, and binsize = 0.0.

neighbor command

Syntax:

```
neighbor skin style
```

- skin = extra distance beyond force cutoff (distance units)
- style = *bin* or *nsq* or *multi*

Examples:

```
neighbor 0.3 bin
neighbor 2.0 nsq
```

Description:

This command sets parameters that affect the building of pairwise neighbor lists. All atom pairs within a neighbor cutoff distance equal to the their force cutoff plus the *skin* distance are stored in the list. Typically, the larger the skin distance, the less often neighbor lists need to be built, but more pairs must be checked for possible force interactions every timestep. The default value for *skin* depends on the choice of units for the simulation; see the default values below.

The *skin* distance is also used to determine how often atoms migrate to new processors if the *check* option of the neigh_modify command is set to *yes*. Atoms are migrated (communicated) to new processors on the same timestep that neighbor lists are re−built.

The *style* value selects what algorithm is used to build the list. The *bin* style creates the list by binning which is an operation that scales linearly with N/P, the number of atoms per processor where N = total number of atoms and P = number of processors. It is almost always faster than the *nsq* style which scales as (N/P)^2. For unsolvated small molecules in a non−periodic box, the *nsq* choice can sometimes be faster. Either style should give the same answers.

The *multi* style is a modified binning algorithm that is useful for systems with a wide range of cutoff distances, e.g. due to different size particles. For the *bin* style, the bin size is set to 1/2 of the largest cutoff distance between any pair of atom types and a single set of bins is defined to search over for all atom types. This can be inefficient if one pair of types has a very long cutoff, but other type pairs have a much shorter cutoff. For style *multi* the bin size is set to 1/2 of the shortest cutoff distance and multiple sets of bins are defined to search over for different atom types. This imposes some extra setup overhead, but the searches themselves may be much faster for the short−cutoff cases. See the communicate multi command for a communication option option that may also be beneficial for simulations of this kind.

The neigh_modify command has additional options that control how often neighbor lists are built and which pairs are stored in the list.

When a run is finished, counts of the number of neighbors stored in the pairwise list and the number of times neighbor lists were built are printed to the screen and log file. See this section for details.

Restrictions: none

Related commands:

neigh_modify, units, communicate

Default:

0.3 bin for units = lj, skin = 0.3 sigma
2.0 bin for units = real or metal, skin = 2.0 Angstroms
0.001 bin for units = si, skin = 0.001 meters = 1.0 mm
0.1 bin for units = cgs, skin = 0.1 cm = 1.0 mm

newton command

Syntax:

```
newton flag
newton flag1 flag2
```

- flag = *on* or *off* for both pairwise and bonded interactions
- flag1 = *on* or *off* for pairwise interactions
- flag2 = *on* or *off* for bonded interactions

Examples:

```
newton off
newton on off
```

Description:

This command turns Newton's 3rd law *on* or *off* for pairwise and bonded interactions. For most problems, setting Newton's 3rd law to *on* means a modest savings in computation at the cost of two times more communication. Whether this is faster depends on problem size, force cutoff lengths, a machine's compute/communication ratio, and how many processors are being used.

Setting the pairwise newton flag to *off* means that if two interacting atoms are on different processors, both processors compute their interaction and the resulting force information is not communicated. Similarly, for bonded interactions, newton *off* means that if a bond, angle, dihedral, or improper interaction contains atoms on 2 or more processors, the interaction is computed by each processor.

LAMMPS should produce the same answers for any newton flag settings, except for round−off issues.

With run_style *respa* and only bonded interactions (bond, angle, etc) computed in the innermost timestep, it may be faster to turn newton *off* for bonded interactions, to avoid extra communication in the innermost loop.

Restrictions:

The newton bond setting cannot be changed after the simulation box is defined by a read_data or create_box command.

Related commands:

run_style respa

Default:

```
newton on
```

next command

Syntax:

```
next variables
```

- variables = one or more variable names

Examples:

```
next x
next a t x myTemp
```

Description:

This command is used with variables defined by the variable command. It assigns the next value to the variable from the list of values defined for that variable by the variable command. Thus when that variable is subsequently substituted for in an input script command, the new value is used.

See the variable command for info on how to define and use different kinds of variables in LAMMPS input scripts. If a variable name is a single lower−case character from "a" to "z", it can be used in an input script command as $a or $z. If it is multiple letters, it can be used as ${myTemp}.

If multiple variables are used as arguments to the *next* command, then all must be of the same variable style: *index*, *loop*, *universe*, or *uloop*. An exception is that *universe*− and *uloop*−style variables can be mixed in the same *next* command.

All the variables specified with the next command are incremented by one value from their respective list or values. *String*− or *atom*− or *equal*− or *world*−style variables cannot be used with the the next command, since they only store a single value.

When any of the variables in the next command has no more values, a flag is set that causes the input script to skip the next jump command encountered. This enables a loop containing a next command to exit. As explained in the variable command, the variable that has exhausted its values is also deleted. This allows it to be used and re−defined later in the input script.

When the next command is used with *index*− or *loop*−style variables, the next value is assigned to the variable for all processors. When the next command is used with *universe*− or *uloop*−style variables, the next value is assigned to whichever processor partition executes the command first. All processors in the partition are assigned the same value. Running LAMMPS on multiple partitions of processors via the "−partition" command−line switch is described in this section of the manual. *Universe*− and *uloop*−style variables are incremented using the files "tmp.lammps.variable" and "tmp.lammps.variable.lock" which you will see in your directory during such a LAMMPS run.

Here is an example of running a series of simulations using the next command with an *index*−style variable. If this input script is named in.polymer, 8 simulations would be run using data files from directories run1 thru run8.

```
variable d index run1 run2 run3 run4 run5 run6 run7 run8
shell cd $d
read_data data.polymer
run 10000
shell cd ..
```

```
clear
next d
jump in.polymer
```

If the variable "d" were of style *universe*, and the same in.polymer input script were run on 3 partitions of processors, then the first 3 simulations would begin, one on each set of processors. Whichever partition finished first, it would assign variable "d" the 4th value and run another simulation, and so forth until all 8 simulations were finished.

Jump and next commands can also be nested to enable multi−level loops. For example, this script will run 15 simulations in a double loop.

```
variable i loop 3
  variable j loop 5
  clear
  ...
  read_data data.polymer.$i$j
  print Running simulation $i.$j
  run 10000
  next j
  jump in.script
next i
jump in.script
```

Here is an example of a double loop which uses the if and jump commands to break out of the inner loop when a condition is met, then continues iterating thru the outer loop.

```
label       loopa
variable    a loop 5
  label     loopb
  variable  b loop 5
  print     "A,B = $a,$b"
  run       10000
  if        $b > 2 then "jump in.script break"
  next      b
  jump      in.script loopb
label       break
variable    b delete

next        a
jump        in.script loopa
```

Restrictions: none

Related commands:

jump, include, shell, variable,

Default: none

orient command

Syntax:

```
orient dim i j k
```

- dim = *x* or *y* or *z*
- i,j,k = orientation of lattice that is along box direction dim

Examples:

```
orient x 1 1 0
orient y −1 1 0
orient z 0 0 1
```

Description:

Specify the orientation of a cubic lattice along simulation box directions *x* or *y* or *z*. These 3 basis vectors are used when the create_atoms command generates a lattice of atoms.

The 3 basis vectors B1, B2, B3 must be mutually orthogonal and form a right−handed system such that B1 cross B2 is in the direction of B3.

The basis vectors should be specified in an irreducible form (smallest possible integers), though LAMMPS does not check for this.

Restrictions: none

Related commands:

origin, create_atoms

Default:

```
orient x 1 0 0
orient y 0 1 0
orient z 0 0 1
```

origin command

Syntax:

```
origin x y z
```

- x,y,z = origin of a lattice

Examples:

```
origin 0.0 0.5 0.5
```

Description:

Set the origin of the lattice defined by the lattice command. The lattice is used by the create_atoms command to create new atoms and by other commands that use a lattice spacing as a distance measure. This command offsets the origin of the lattice from the (0,0,0) coordinate of the simulation box by some fraction of a lattice spacing in each dimension.

The specified values are in lattice coordinates from 0.0 to 1.0, so that a value of 0.5 means the lattice is displaced 1/2 a cubic cell.

Restrictions: none

Related commands:

lattice, orient

Default:

```
origin 0 0 0
```

package command

Syntax:

```
package style args
```

- style = *gpu* or *cuda* or *omp*
- args = arguments specific to the style

```
  gpu args = mode first last split
      mode = force or force/neigh
      first = ID of first GPU to be used on each node
      last = ID of last GPU to be used on each node
      split = fraction of particles assigned to the GPU
    cuda args = to be determined
    omp args = Nthreads
      Nthreads = # of OpenMP threads to associate with each MPI process
```

Examples:

```
package gpu force 0 0 1.0
package gpu force 0 0 0.75
package gpu force/neigh 0 0 1.0
package gpu force/neigh 0 1 −1.0
package cuda blah
package omp 4
```

Description:

This command invokes package−specific settings. Currently the following packages use it: GPU, USER−CUDA, and USER−OMP.

See this section of the manual for more details about using these various packages for accelerating a LAMMPS calculation.

The *gpu* style invokes options associated with the use of the GPU package. It allows you to select and initialize GPUs to be used for acceleration via this package and configure how the GPU acceleration is performed. These settings are required in order to use any style with GPU acceleration.

The *mode* setting specifies where neighbor list calculations will be performed. If *mode* is force, neighbor list calculation is performed on the CPU. If *mode* is force/neigh, neighbor list calculation is performed on the GPU. GPU neighbor list calculation currently cannot be used with a triclinic box. GPU neighbor list calculation currently cannot be used with hybrid pair styles. GPU neighbor lists are not compatible with styles that are not GPU−enabled. When a non−GPU enabled style requires a neighbor list, it will also be built using CPU routines. In these cases, it will typically be more efficient to only use CPU neighbor list builds.

The *first* and *last* settings specify the GPUs that will be used for simulation. On each node, the GPU IDs in the inclusive range from *first* to *last* will be used.

The *split* setting can be used for load balancing force calculation work between CPU and GPU cores in GPU−enabled pair styles. If 0 < *split* < 1.0, a fixed fraction of particles is offloaded to the GPU while force calculation for the other particles occurs simulataneously on the CPU. If *split*split = 1.0, all force calculations for GPU accelerated pair styles are performed on the GPU. In this case, hybrid, bond, angle, dihedral, improper, and

long−range calculations can be performed on the CPU while the GPU is performing force calculations for the GPU−enabled pair style. If all CPU force computations complete before the GPU, LAMMPS will block until the GPU has finished before continuing the timestep.

As an example, if you have two GPUs per node and 8 CPU cores per node, and would like to run on 4 nodes (32 cores) with dynamic balancing of force calculation across CPU and GPU cores, you could specify

```
package gpu force/neigh 0 1 −1
```

In this case, all CPU cores and GPU devices on the nodes would be utilized. Each GPU device would be shared by 4 CPU cores. The CPU cores would perform force calculations for some fraction of the particles at the same time the GPUs performed force calculation for the other particles.

The *cuda* style invokes options associated with the use of the USER−CUDA package. These still need to be documented.

The *omp* style invokes options associated with the use of the USER−OMP package.

The only setting to make is the number of OpenMP threads to be allocated for each MPI process. For example, if your system has nodes with dual quad−core processors, it has a total of 8 cores per node. You could run MPI on 2 cores on each node (e.g. using options for the mpirun command), and set the *Nthreads* setting to 4. This would effectively use all 8 cores on each node. Since each MPI process would spawn 4 threads (one of which runs as part of the MPI process itself).

For performance reasons, you should not set *Nthreads* to more threads than there are physical cores, but LAMMPS does not check for this.

Restrictions:

This command cannot be used after the simulation box is defined by a read_data or create_box command.

The cuda style of this command can only be invoked if LAMMPS was built with the USER−CUDA package. See the Making LAMMPS section for more info.

The gpu style of this command can only be invoked if LAMMPS was built with the GPU package. See the Making LAMMPS section for more info.

The omp style of this command can only be invoked if LAMMPS was built with the USER−OMP package. See the Making LAMMPS section for more info.

Related commands: none

Default: none

pair_style adp command

Syntax:

```
pair_style adp
```

Examples:

```
pair_style adp
pair_coeff * * Ta.adp Ta
pair_coeff * * ../potentials/AlCu.adp Al Al Cu
```

Description:

Style *adp* computes pairwise interactions for metals and metal alloys using the angular dependent potential (ADP) of (Mishin), which is a generalization of the embedded atom method (EAM) potential. The LAMMPS implementation is discussed in (Singh). The total energy Ei of an atom I is given by

$$
\begin{aligned}
E_i &= F_\alpha \left(\sum_{j \neq i} \rho(r_{ij}) \right) + \frac{1}{2} \sum_{j \neq i} \phi_{\alpha\beta}(r_{ij}) + \frac{1}{2} \sum_{i,s} (\mu_i^s)^2 + \sum_{i,s,t} \left(\lambda_i^{st} \right)^2 - \frac{1}{6} \nu_i \\
\mu_i^s &= \sum_{i \neq j} u(r_{ij}) r_{ij}^s \\
\lambda_i^{st} &= \sum_{i \neq j} w(r_{ij}) r_{ij}^s r_{ij}^t \\
\nu_i &= \sum_s \lambda_i^{ss}
\end{aligned}
$$

where F is the embedding energy which is a function of the atomic electron density rho, phi is a pair potential interaction, alpha and beta are the element types of atoms I and J, and s and t = 1,2,3 and refer to the cartesian coordinates. The mu and lambda terms represent the dipole and quadruple distortions of the local atomic environment which extend the original EAM framework by introducing angular forces.

Note that unlike for other potentials, cutoffs for ADP potentials are not set in the pair_style or pair_coeff command; they are specified in the ADP potential files themselves. Likewise, the ADP potential files list atomic masses; thus you do not need to use the mass command to specify them.

The NIST WWW site distributes and documents ADP potentials:

```
http://www.ctcms.nist.gov/potentials
```

Note that these must be converted into the extended DYNAMO *setfl* format discussed below.

The NIST site is maintained by Chandler Becker (cbecker at nist.gov) who is good resource for info on interatomic potentials and file formats.

Only a single pair_coeff command is used with the *adp* style which specifies an extended DYNAMO *setfl* file, which contains information for M elements. These are mapped to LAMMPS atom types by specifying N additional arguments after the filename in the pair_coeff command, where N is the number of LAMMPS atom

types:

- filename
- N element names = mapping of extended *setfl* elements to atom types

As an example, the potentials/AlCu.adp file, included in the potentials directory of the LAMMPS distrbution, is an extended *setfl* file which has tabulated ADP values for w elements and their alloy interactions: Cu and Al. If your LAMMPS simulation has 4 atoms types and you want the 1st 3 to be Al, and the 4th to be Cu, you would use the following pair_coeff command:

```
pair_coeff * * AlCu.adp Al Al Al Cu
```

The 1st 2 arguments must be * * so as to span all LAMMPS atom types. The first three Al arguments map LAMMPS atom types 1,2,3 to the Al element in the extended *setfl* file. The final Cu argument maps LAMMPS atom type 4 to the Al element in the extended *setfl* file. Note that there is no requirement that your simulation use all the elements specified by the extended *setfl* file.

If a mapping value is specified as NULL, the mapping is not performed. This can be used when an *adp* potential is used as part of the *hybrid* pair style. The NULL values are placeholders for atom types that will be used with other potentials.

Adp files in the *potentials* directory of the LAMMPS distribution have an ".adp" suffix. A DYNAMO *setfl* file extended for ADP is formatted as follows. Basically it is the standard *setfl* format with additional tabulated functions u and w added to the file after the tabulated pair potentials. See the pair_eam command for further details on the *setfl* format.

- lines 1,2,3 = comments (ignored)
- line 4: Nelements Element1 Element2 ... ElementN
- line 5: Nrho, drho, Nr, dr, cutoff

Following the 5 header lines are Nelements sections, one for each element, each with the following format:

- line 1 = atomic number, mass, lattice constant, lattice type (e.g. FCC)
- embedding function F(rho) (Nrho values)
- density function rho(r) (Nr values)

Following the Nelements sections, Nr values for each pair potential phi(r) array are listed for all i,j element pairs in the same format as other arrays. Since these interactions are symmetric (i,j = j,i) only phi arrays with i >= j are listed, in the following order: i,j = (1,1), (2,1), (2,2), (3,1), (3,2), (3,3), (4,1), ..., (Nelements, Nelements).

After the phi(r) arrays, each of the u(r) arrays are listed in the same order with the same assumptions of symmetry. Directly following the u(r), the w(r) arrays are listed.

Mixing, shift, table, tail correction, restart, rRESPA info:

For atom type pairs I,J and I != J, where types I and J correspond to two different element types, no special mixing rules are needed, since the ADP potential files specify alloy interactions explicitly.

This pair style does not support the pair_modify shift, table, and tail options.

This pair style does not write its information to binary restart files, since it is stored in tabulated potential files. Thus, you need to re−specify the pair_style and pair_coeff commands in an input script that reads a restart file.

This pair style can only be used via the *pair* keyword of the run_style respa command. It does not support the *inner*, *middle*, *outer* keywords.

Restrictions:

This pair style is part of the "manybody" package. It is only enabled if LAMMPS was built with that package (which it is by default).

Related commands:

pair_coeff, pair_eam

Default: none

(Mishin) Mishin, Mehl, and Papaconstantopoulos, Acta Mater, 53, 4029 (2005).

(Singh) Singh and Warner, Acta Mater, 58, 5797−5805 (2010),

pair_style airebo command

pair_style rebo command

Syntax:

```
pair_style style cutoff LJ_flag TORSION_flag
```

- style = *airebo* or *rebo*
- cutoff = LJ cutoff (sigma scale factor) (AIREBO only)
- LJ_flag = 0/1 to turn off/on the LJ term (AIREBO only, optional)
- TORSION_flag = 0/1 to turn off/on the torsion term (AIREBO only, optional)

Examples:

```
pair_style airebo 3.0
pair_style airebo 2.5 1 0
pair_coeff * * ../potentials/CH.airebo H C

pair_style rebo
pair_coeff * * ../potentials/CH.airebo H C
```

Description:

The *airebo* pair style computes the Adaptive Intermolecular Reactive Empirical Bond Order (AIREBO) Potential of (Stuart) for a system of carbon and/or hydrogen atoms. Note that this is the initial formulation of AIREBO from 2000, not the later formulation. The *rebo* pair style computes the Reactive Empirical Bond Order (REBO) Potential of (Brenner). Note that this is the so−called 2nd generation REBO from 2002, not the original REBO from 1990. As discussed below, 2nd generation REBO is closely related to the intial AIREBO; it is just a subset of the potential energy terms.

The AIREBO potential consists of three terms:

$$E = \frac{1}{2}\sum_i \sum_{j \neq i} \left[E_{ij}^{REBO} + E_{ij}^{LJ} + \sum_{k \neq i,j} \sum_{l \neq i,j,k} E_{kijl}^{TORSION} \right]$$

By default, all three terms are included. For the *airebo* style, if the two optional flag arguments to the pair_style command are included, the LJ and torsional terms can be turned off. Note that both or neither of the flags must be included. If both of the LJ an torsional terms are turned off, it becomes the 2nd−generation REBO potential, with a small caveat on the spline fitting procedure mentioned below. This can be specified directly as pair_style rebo with no additional arguments.

The detailed formulas for this potential are given in (Stuart); here we provide only a brief description.

The E_REBO term has the same functional form as the hydrocarbon REBO potential developed in (Brenner). The coefficients for E_REBO in AIREBO are essentially the same as Brenner's potential, but a few fitted spline values are slightly different. For most cases the E_REBO term in AIREBO will produce the same energies, forces and statistical averages as the original REBO potential from which it was derived. The E_REBO term in the AIREBO potential gives the model its reactive capabilities and only describes short−ranged C−C, C−H and H−H

interactions (r < 2 Angstroms). These interactions have strong coordination−dependence through a bond order parameter, which adjusts the attraction between the I,J atoms based on the position of other nearby atoms and thus has 3− and 4−body dependence.

The E_LJ term adds longer−ranged interactions (2 < r < cutoff) using a form similar to the standard Lennard Jones potential. The E_LJ term in AIREBO contains a series of switching functions so that the short−ranged LJ repulsion (1/r^12) does not interfere with the energetics captured by the E_REBO term. The extent of the E_LJ interactions is determined by the *cutoff* argument to the pair_style command which is a scale factor. For each type pair (C−C, C−H, H−H) the cutoff is obtained by multiplying the scale factor by the sigma value defined in the potential file for that type pair. In the standard AIREBO potential, sigma_CC = 3.4 Angstroms, so with a scale factor of 3.0 (the argument in pair_style), the resulting E_LJ cutoff would be 10.2 Angstroms.

The E_TORSION term is an explicit 4−body potential that describes various dihedral angle preferences in hydrocarbon configurations.

Only a single pair_coeff command is used with the *airebo* or *rebo* style which specifies an AIREBO potential file with parameters for C and H. Note that the *rebo* style in LAMMPS uses the same AIREBO−formatted potential file. These are mapped to LAMMPS atom types by specifying N additional arguments after the filename in the pair_coeff command, where N is the number of LAMMPS atom types:

- filename
- N element names = mapping of AIREBO elements to atom types

As an example, if your LAMMPS simulation has 4 atom types and you want the 1st 3 to be C, and the 4th to be H, you would use the following pair_coeff command:

```
pair_coeff * * CH.airebo C C C H
```

The 1st 2 arguments must be * * so as to span all LAMMPS atom types. The first three C arguments map LAMMPS atom types 1,2,3 to the C element in the AIREBO file. The final H argument maps LAMMPS atom type 4 to the H element in the SW file. If a mapping value is specified as NULL, the mapping is not performed. This can be used when a *airebo* potential is used as part of the *hybrid* pair style. The NULL values are placeholders for atom types that will be used with other potentials.

The parameters/coefficients for the AIREBO potentials are listed in the CH.airebo file to agree with the original (Stuart) paper. Thus the parameters are specific to this potential and the way it was fit, so modifying the file should be done cautiously.

Mixing, shift, table, tail correction, restart, rRESPA info:

These pair styles do not support the pair_modify mix, shift, table, and tail options.

These pair styles do not write their information to binary restart files, since it is stored in potential files. Thus, you need to re−specify the pair_style and pair_coeff commands in an input script that reads a restart file.

These pair styles can only be used via the *pair* keyword of the run_style respa command. They do not support the *inner*, *middle*, *outer* keywords.

Restrictions:

These pair styles are part of the "manybody" package. They are only enabled if LAMMPS was built with that package (which it is by default). See the Making LAMMPS section for more info.

These pair potentials require the newton setting to be "on" for pair interactions.

The CH.airebo potential file provided with LAMMPS (see the potentials directory) is parameterized for metal units. You can use the AIREBO or REBO potential with any LAMMPS units, but you would need to create your own AIREBO potential file with coefficients listed in the appropriate units if your simulation doesn't use "metal" units.

Related commands:

pair_coeff

Default: none

(Stuart) Stuart, Tutein, Harrison, J Chem Phys, 112, 6472−6486 (2000).

(Brenner) Brenner, Shenderova, Harrison, Stuart, Ni, Sinnott, J Physics: Condensed Matter, 14, 783−802 (2002).

pair_style awpmd/cut command

Syntax:

```
pair_style awpmd/cut Rc keyword value ...
```

- Rc = global cutoff, −1 means cutoff of half the shortest box length
- zero or more keyword/value pairs may be appended
- keyword = *hartree* or *dproduct* or *uhf* or *free* or *pbc* or *fix* or *harm* or *ermscale* or *flex_press*

```
hartree value = none
  dproduct value = none
  uhf value = none
  free value = none
  pbc value = Plen
    Plen = periodic width of electron = −1 or positive value (distance units)
  fix value = Flen
    Flen = fixed width of electron = −1 or positive value (distance units)
  harm value = width
    width = harmonic width constraint
  ermscale value = factor
    factor = scaling between electron mass and width variable mass
  flex_press value = none
```

Examples:

```
pair_style awpmd/cut −1
pair_style awpmd/cut 40.0 uhf free
pair_coeff * *
pair_coeff 2 2 20.0
```

Description:

This pair style contains an implementation of the Antisymmetrized Wave Packet Molecular Dynamics (AWPMD) method. Need citation here. Need basic formulas here. Could be links to other documents.

Rc is the cutoff.

The pair_style command allows for several optional keywords to be specified.

The *hartree*, *dproduct*, and *uhf* keywords specify the form of the initial trial wave function for the system. If the *hartree* keyword is used, then a Hartree multielectron trial wave function is used. If the *dproduct* keyword is used, then a trial function which is a product of two determinants for each spin type is used. If the *uhf* keyword is used, then an unrestricted Hartree−Fock trial wave function is used.

The *free*, *pbc*, and *fix* keywords specify a width constraint on the electron wavepackets. If the *free* keyword is specified, then there is no constraint. If the *pbc* keyword is used and *Plen* is specified as −1, then the maximum width is half the shortest box length. If *Plen* is a positive value, then the value is the maximum width. If the *fix* keyword is used and *Flen* is specified as −1, then electrons have a constant width that is read from the data file. If *Flen* is a positive value, then the constant width for all electrons is set to *Flen*.

The *harm* keyword allow oscillations in the width of the electron wavepackets. More details are needed.

The *ermscale* keyword specifies a unitless scaling factor between the electron masses and the width variable mass. More details needed.

If the *flex_press* keyword is used, then a contribution from the electrons is added to the total virial and pressure of the system.

This potential is designed to be used with atom_style wavepacket definitions, in order to handle the description of systems with interacting nuclei and explicit electrons.

The following coefficients must be defined for each pair of atoms types via the pair_coeff command as in the examples above, or in the data file or restart files read by the read_data or read_restart commands, or by mixing as described below:

- cutoff (distance units)

For *awpmd/cut*, the cutoff coefficient is optional. If it is not used (as in some of the examples above), the default global value specified in the pair_style command is used.

Mixing, shift, table, tail correction, restart, rRESPA info:

The pair_modify mix, shift, table, and tail options are not relevant for this pair style.

This pair style writes its information to binary restart files, so pair_style and pair_coeff commands do not need to be specified in an input script that reads a restart file.

This pair style can only be used via the *pair* keyword of the run_style respa command. It does not support the *inner*, *middle*, *outer* keywords.

Restrictions: none

Related commands:

pair_coeff

Default:

These are the defaults for the pair_style keywords: *hartree* for the initial wavefunction, *free* for the wavepacket width.

pair_style born command

pair_style born/coul/long command

pair_style born/coul/long/cuda command

Syntax:

```
pair_style style args
```

- style = *born* or *born/coul/long*
- args = list of arguments for a particular style

```
born args = cutoff
    cutoff = global cutoff for non-Coulombic interactions (distance units)
  born/coul/long args = cutoff (cutoff2)
    cutoff = global cutoff for non-Coulombic (and Coulombic if only 1 arg) (distance units)
    cutoff2 = global cutoff for Coulombic (optional) (distance units)
```

Examples:

```
pair_style born 10.0
pair_coeff * * 6.08 0.317 2.340 24.18 11.51
pair_coeff 1 1 6.08 0.317 2.340 24.18 11.51

pair_style born/coul/long 10.0
pair_style born/coul/long 10.0 8.0
pair_coeff * * 6.08 0.317 2.340 24.18 11.51
pair_coeff 1 1 6.08 0.317 2.340 24.18 11.51
```

Description:

The *born* style computes the Born−Mayer−Huggins or Tosi/Fumi potential described in (Fumi and Tosi), given by

$$E = A \exp\left(\frac{\sigma - r}{\rho}\right) - \frac{C}{r^6} + \frac{D}{r^8} \qquad r < r_c$$

where sigma is an interaction−dependent length parameter, rho is an ionic−pair dependent length parameter, and Rc is the cutoff.

The *born/coul/long* style adds a Coulombic term as described for the lj/cut pair styles. An additional damping factor is applied to the Coulombic term so it can be used in conjunction with the kspace_style command and its *ewald* or *pppm* option. The Coulombic cutoff specified for this style means that pairwise interactions within this distance are computed directly; interactions outside that distance are computed in reciprocal space.

If one cutoff is specified for the *born/coulk/long* style, it is used for both the A,C,D and Coulombic terms. If two cutoffs are specified, the first is used as the cutoff for the A,C,D terms, and the second is the cutoff for the Coulombic term.

Note that these potentials are related to the Buckingham potential.

The following coefficients must be defined for each pair of atoms types via the pair_coeff command as in the examples above, or in the data file or restart files read by the read_data or read_restart commands, or by mixing as described below:

- A (energy units)
- rho (distance units)
- sigma (distance units)
- C (energy units * distance units^6)
- D (energy units * distance units^8)
- cutoff (distance units)

The second coefficient, rho, must be greater than zero.

The last coefficient is optional. If not specified, the global A,C,D cutoff specified in the pair_style command is used.

For *buck/coul/long* no Coulombic cutoff can be specified for an individual I,J type pair. All type pairs use the same global Coulombic cutoff specified in the pair_style command.

Styles with a *cuda*, *gpu*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in this section of the manual. The accelerated styles take the same arguments and should produce the same results, except for round−off and precision issues.

These accelerated styles are part of the "user−cuda", "gpu", and "opt" packages respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the −suffix command−line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See this section of the manual for more instructions on how to use the accelerated styles effectively.

Mixing, shift, table, tail correction, restart, rRESPA info:

These pair styles do not support mixing. Thus, coefficients for all I,J pairs must be specified explicitly.

These styles support the pair_modify shift option for the energy of the exp(), 1/r^6, and 1/r^8 portion of the pair interaction.

The *born/coul/long* pair style does not support the pair_modify table option since a tabulation capability has not yet been added to this potential.

These styles support the pair_modify tail option for adding long−range tail corrections to energy and pressure.

Thess styles writes thei information to binary restart files, so pair_style and pair_coeff commands do not need to be specified in an input script that reads a restart file.

These styles can only be used via the *pair* keyword of the run_style respa command. They do not support the *inner*, *middle*, *outer* keywords.

Restrictions:

The *born/coul/long* style is part of the "kspace" package. It is only enabled if LAMMPS was built with that package (which it is by default). See the Making LAMMPS section for more info.

Related commands:

pair_coeff, pair_style buck

Default: none

Fumi and Tosi, J Phys Chem Solids, 25, 31 (1964), Fumi and Tosi, J Phys Chem Solids, 25, 45 (1964).

pair_style buck command

pair_style buck/cuda command

pair_style buck/coul/cut command

pair_style buck/coul/cut/cuda command

pair_style buck/coul/long command

pair_style buck/coul/long/cuda command

Syntax:

```
pair_style style args
```

- style = *buck* or *buck/coul/cut* or *buck/coul/long*
- args = list of arguments for a particular style

```
buck args = cutoff
    cutoff = global cutoff for Buckingham interactions (distance units)
  buck/coul/cut args = cutoff (cutoff2)
    cutoff = global cutoff for Buckingham (and Coulombic if only 1 arg) (distance units)
    cutoff2 = global cutoff for Coulombic (optional) (distance units)
  buck/coul/long args = cutoff (cutoff2)
    cutoff = global cutoff for Buckingham (and Coulombic if only 1 arg) (distance units)
    cutoff2 = global cutoff for Coulombic (optional) (distance units)
```

Examples:

```
pair_style buck 2.5
pair_coeff * * 100.0 1.5 200.0
pair_coeff * * 100.0 1.5 200.0 3.0

pair_style buck/coul/cut 10.0
pair_style buck/coul/cut 10.0 8.0
pair_coeff * * 100.0 1.5 200.0
pair_coeff 1 1 100.0 1.5 200.0 9.0
pair_coeff 1 1 100.0 1.5 200.0 9.0 8.0

pair_style buck/coul/long 10.0
pair_style buck/coul/long 10.0 8.0
pair_coeff * * 100.0 1.5 200.0
pair_coeff 1 1 100.0 1.5 200.0 9.0
```

Description:

The *buck* style computes a Buckingham potential (exp/6 instead of Lennard−Jones 12/6) given by

$$E = Ae^{-r/\rho} - \frac{C}{r^6} \qquad r < r_c$$

where rho is an ionic−pair dependent length parameter, and Rc is the cutoff.

The *buck/coul/cut* and *buck/coul/long* styles add a Coulombic term as described for the lj/cut pair styles. For *buck/coul/long*, an additional damping factor is applied to the Coulombic term so it can be used in conjunction with the kspace_style command and its *ewald* or *pppm* option. The Coulombic cutoff specified for this style means that pairwise interactions within this distance are computed directly; interactions outside that distance are computed in reciprocal space.

If one cutoff is specified for the *born/coul/cut* and *born/coulk/long* styles, it is used for both the A,C and Coulombic terms. If two cutoffs are specified, the first is used as the cutoff for the A,C terms, and the second is the cutoff for the Coulombic term.

Note that these potentials are related to the Born−Mayer−Huggins potential.

The following coefficients must be defined for each pair of atoms types via the pair_coeff command as in the examples above, or in the data file or restart files read by the read_data or read_restart commands:

- A (energy units)
- rho (distance units)
- C (energy−distance^6 units)
- cutoff (distance units)
- cutoff2 (distance units)

The second coefficient, rho, must be greater than zero.

The latter 2 coefficients are optional. If not specified, the global A,C and Coulombic cutoffs are used. If only one cutoff is specified, it is used as the cutoff for both A,C and Coulombic interactions for this type pair. If both coefficients are specified, they are used as the A,C and Coulombic cutoffs for this type pair. You cannot specify 2 cutoffs for style *buck*, since it has no Coulombic terms.

For *buck/coul/long* only the LJ cutoff can be specified since a Coulombic cutoff cannot be specified for an individual I,J type pair. All type pairs use the same global Coulombic cutoff specified in the pair_style command.

Styles with a *cuda*, *gpu*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in this section of the manual. The accelerated styles take the same arguments and should produce the same results, except for round−off and precision issues.

These accelerated styles are part of the "user−cuda", "gpu", and "opt" packages respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the −suffix command−line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See this section of the manual for more instructions on how to use the accelerated styles effectively.

Mixing, shift, table, tail correction, restart, rRESPA info:

These pair styles do not support mixing. Thus, coefficients for all I,J pairs must be specified explicitly.

These styles support the pair_modify shift option for the energy of the exp() and 1/r^6 portion of the pair interaction.

The *buck/coul/long* pair style does not support the pair_modify table option since a tabulation capability has not yet been added to this potential.

These styles support the pair_modify tail option for adding long−range tail corrections to energy and pressure for the A,C terms in the pair interaction.

These styles write their information to binary restart files, so pair_style and pair_coeff commands do not need to be specified in an input script that reads a restart file.

These styles can only be used via the *pair* keyword of the run_style respa command. They do not support the *inner*, *middle*, *outer* keywords.

Restrictions:

The *buck/coul/long* style is part of the "kspace" package. It is only enabled if LAMMPS was built with that package (which it is by default). See the Making LAMMPS section for more info.

Related commands:

pair_coeff, pair_style born

Default: none

pair_style buck/coul command

Syntax:

```
pair_style buck/coul flag_buck flag_coul cutoff (cutoff2)
```

- flag_buck = *long* or *cut*

```
  long = use Kspace long-range summation for the dispersion term 1/r^6
    cut = use a cutoff
```

- flag_coul = *long* or *off*

```
  long = use Kspace long-range summation for the Coulombic term 1/r
    off = omit the Coulombic term
```

- cutoff = global cutoff for Buckingham (and Coulombic if only 1 cutoff) (distance units)
- cutoff2 = global cutoff for Coulombic (optional) (distance units)

Examples:

```
pair_style buck/coul cut off 2.5
pair_style buck/coul cut long 2.5 4.0
pair_style buck/coul long long 2.5 4.0
pair_coeff * * 1 1
pair_coeff 1 1 1 3 4
```

Description:

The *buck/coul* style computes a Buckingham potential (exp/6 instead of Lennard−Jones 12/6) and Coulombic potential, given by

$$E = Ae^{-r/\rho} - \frac{C}{r^6} \qquad r < r_c$$

$$E = \frac{Cq_iq_j}{\epsilon r} \qquad r < r_c$$

Rc is the cutoff. If one cutoff is specified in the pair_style command, it is used for both the Buckingham and Coulombic terms. If two cutoffs are specified, they are used as cutoffs for the Buckingham and Coulombic terms respectively.

The purpose of this pair style is to capture long−range interactions resulting from both attractive 1/r^6 Buckingham and Coulombic 1/r interactions. This is done by use of the *flag_buck* and *flag_coul* settings. The "Ismail paper has more details on when it is appropriate to include long−range 1/r^6 interactions, using this potential.

If *flag_buck* is set to *long*, no cutoff is used on the Buckingham 1/r^6 dispersion term. The long−range portion is calculated by using the kspace_style ewald/n command. The specified Buckingham cutoff then determines which portion of the Buckingham interactions are computed directly by the pair potential versus which part is computed in reciprocal space via the Kspace style. If *flag_buck* is set to *cut*, the Buckingham interactions are simply cutoff,

as with pair_style buck.

If *flag_coul* is set to *long*, no cutoff is used on the Coulombic interactions. The long−range portion is calculated by using any style, including *ewald/n* of the kspace_style command. Note that if *flag_buck* is also set to long, then only the *ewald/n* Kspace style can perform the long−range calculations for both the Buckingham and Coulombic interactions. If *flag_coul* is set to *off*, Coulombic interactions are not computed.

The following coefficients must be defined for each pair of atoms types via the pair_coeff command as in the examples above, or in the data file or restart files read by the read_data or read_restart commands:

- A (energy units)
- rho (distance units)
- C (energy−distance^6 units)
- cutoff (distance units)
- cutoff2 (distance units)

The second coefficient, rho, must be greater than zero.

The latter 2 coefficients are optional. If not specified, the global Buckingham and Coulombic cutoffs specified in the pair_style command are used. If only one cutoff is specified, it is used as the cutoff for both Buckingham and Coulombic interactions for this type pair. If both coefficients are specified, they are used as the Buckingham and Coulombic cutoffs for this type pair. Note that if you are using *flag_buck* set to *long*, you cannot specify a Buckingham cutoff for an atom type pair, since only one global Buckingham cutoff is allowed. Similarly, if you are using *flag_coul* set to *long*, you cannot specify a Coulombic cutoff for an atom type pair, since only one global Coulombic cutoff is allowed.

Mixing, shift, table, tail correction, restart, rRESPA info:

This pair styles does not support mixing. Thus, coefficients for all I,J pairs must be specified explicitly.

This pair style supports the pair_modify shift option for the energy of the exp() and 1/r^6 portion of the pair interaction, assuming *flag_buck* is *cut*.

This pair style does not support the pair_modify shift option for the energy of the Buckingham portion of the pair interaction.

This pair style does not support the pair_modify table option since a tabulation capability has not yet been added to this potential.

This pair style write its information to binary restart files, so pair_style and pair_coeff commands do not need to be specified in an input script that reads a restart file.

This pair style supports the use of the *inner*, *middle*, and *outer* keywords of the run_style respa command, meaning the pairwise forces can be partitioned by distance at different levels of the rRESPA hierarchy. See the run_style command for details.

Restrictions:

This style is part of the "user−ewaldn" package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

Related commands:

pair_coeff

Default: none

(Ismail) Ismail, Tsige, In 't Veld, Grest, Molecular Physics (accepted) (2007).

pair_style lj/charmm/coul/charmm command

pair_style lj/charmm/coul/charmm/cuda command

pair_style lj/charmm/coul/charmm/implicit command

pair_style lj/charmm/coul/charmm/implicit/cuda command

pair_style lj/charmm/coul/long command

pair_style lj/charmm/coul/long/cuda command

pair_style lj/charmm/coul/long/gpu command

pair_style lj/charmm/coul/long/opt command

Syntax:

```
pair_style style args
```

- style = *lj/charmm/coul/charmm* or *lj/charmm/coul/charmm/implicit* or *lj/charmm/coul/long*
- args = list of arguments for a particular style

```
lj/charmm/coul/charmm args = inner outer (inner2) (outer2)
    inner, outer = global switching cutoffs for Lennard Jones (and Coulombic if only 2 args)
    inner2, outer2 = global switching cutoffs for Coulombic (optional)
  lj/charmm/coul/charmm/implicit args = inner outer (inner2) (outer2)
    inner, outer = global switching cutoffs for LJ (and Coulombic if only 2 args)
    inner2, outer2 = global switching cutoffs for Coulombic (optional)
  lj/charmm/coul/long args = inner outer (cutoff)
    inner, outer = global switching cutoffs for LJ (and Coulombic if only 2 args)
    cutoff = global cutoff for Coulombic (optional, outer is Coulombic cutoff if only 2 args)
```

Examples:

```
pair_style lj/charmm/coul/charmm 8.0 10.0
pair_style lj/charmm/coul/charmm 8.0 10.0 7.0 9.0
pair_coeff * * 100.0 2.0
pair_coeff 1 1 100.0 2.0 150.0 3.5

pair_style lj/charmm/coul/charmm/implicit 8.0 10.0
pair_style lj/charmm/coul/charmm/implicit 8.0 10.0 7.0 9.0
pair_coeff * * 100.0 2.0
pair_coeff 1 1 100.0 2.0 150.0 3.5

pair_style lj/charmm/coul/long 8.0 10.0
pair_style lj/charmm/coul/long 8.0 10.0 9.0
pair_coeff * * 100.0 2.0
pair_coeff 1 1 100.0 2.0 150.0 3.5
```

Description:

The *lj/charmm* styles compute LJ and Coulombic interactions with an additional switching function S(r) that ramps the energy and force smoothly to zero between an inner and outer cutoff. It is a widely used potential in the CHARMM MD code. See (MacKerell) for a description of the CHARMM force field.

$$
\begin{aligned}
E &= LJ(r) && r < r_{\text{in}} \\
&= S(r) * LJ(r) && r_{\text{in}} < r < r_{\text{out}} \\
&= 0 && r > r_{\text{out}} \\
E &= C(r) && r < r_{\text{in}} \\
&= S(r) * C(r) && r_{\text{in}} < r < r_{\text{out}} \\
&= 0 && r > r_{\text{out}} \\
LJ(r) &= 4\epsilon \left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right] \\
C(r) &= \frac{C q_i q_j}{\epsilon r} \\
S(r) &= \frac{\left[r_{\text{out}}^2 - r^2\right]^2 \left[r_{\text{out}}^2 + 2r^2 - 3r_{\text{in}}^2\right]}{\left[r_{\text{out}}^2 - {r_{\text{in}}}^2\right]^3}
\end{aligned}
$$

Both the LJ and Coulombic terms require an inner and outer cutoff. They can be the same for both formulas or different depending on whether 2 or 4 arguments are used in the pair_style command. In each case, the inner cutoff distance must be less than the outer cutoff. It it typical to make the difference between the 2 cutoffs about 1.0 Angstrom.

Style *lj/charmm/coul/charmm/implicit* computes the same formulas as style *lj/charmm/coul/charmm* except that an additional 1/r term is included in the Coulombic formula. The Coulombic energy thus varies as 1/r^2. This is effectively a distance−dependent dielectric term which is a simple model for an implicit solvent with additional screening. It is designed for use in a simulation of an unsolvated biomolecule (no explicit water molecules).

Style *lj/charmm/coul/long* computes the same formulas as style *lj/charmm/coul/charmm* except that an additional damping factor is applied to the Coulombic term, as in the discussion for pair style *lj/cut/coul/long*. Only one Coulombic cutoff is specified for *lj/charmm/coul/long*; if only 2 arguments are used in the pair_style command, then the outer LJ cutoff is used as the single Coulombic cutoff.

The following coefficients must be defined for each pair of atoms types via the pair_coeff command as in the examples above, or in the data file or restart files read by the read_data or read_restart commands, or by mixing as described below:

- epsilon (energy units)
- sigma (distance units)
- epsilon_14 (energy units)
- sigma_14 (distance units)

Note that sigma is defined in the LJ formula as the zero−crossing distance for the potential, not as the energy minimum at 2^(1/6) sigma.

The latter 2 coefficients are optional. If they are specified, they are used in the LJ formula between 2 atoms of these types which are also first and fourth atoms in any dihedral. No cutoffs are specified because this CHARMM force field does not allow varying cutoffs for individual atom pairs; all pairs use the global cutoff(s) specified in the pair_style command.

Styles with a *cuda*, *gpu*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in this section of the manual. The accelerated styles take the same arguments and should produce the same results, except for round−off and precision issues.

These accelerated styles are part of the "user−cuda", "gpu", and "opt" packages respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the −suffix command−line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See this section of the manual for more instructions on how to use the accelerated styles effectively.

Mixing, shift, table, tail correction, restart, rRESPA info:

For atom type pairs I,J and I != J, the epsilon, sigma, epsilon_14, and sigma_14 coefficients for all of the lj/charmm pair styles can be mixed. The default mix value is *arithmetic* to coincide with the usual settings for the CHARMM force field. See the "pair_modify" command for details.

None of the lj/charmm pair styles support the pair_modify shift option, since the Lennard−Jones portion of the pair interaction is smoothed to 0.0 at the cutoff.

The *lj/charmm/coul/long* style supports the pair_modify table option since it can tabulate the short−range portion of the long−range Coulombic interaction.

None of the lj/charmm pair styles support the pair_modify tail option for adding long−range tail corrections to energy and pressure, since the Lennard−Jones portion of the pair interaction is smoothed to 0.0 at the cutoff.

All of the lj/charmm pair styles write their information to binary restart files, so pair_style and pair_coeff commands do not need to be specified in an input script that reads a restart file.

The lj/charmm/coul/long pair style supports the use of the *inner*, *middle*, and *outer* keywords of the run_style respa command, meaning the pairwise forces can be partitioned by distance at different levels of the rRESPA hierarchy. The other styles only support the *pair* keyword of run_style respa. See the run_style command for details.

Restrictions:

The *lj/charmm/coul/charmm* and *lj/charmm/coul/charmm/implicit* styles are part of the "molecule" package. The *lj/charmm/coul/long* style is part of the "kspace" package. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info. Note that the molecule and kspace packages are installed by default.

Related commands:

pair_coeff

Default: none

(MacKerell) MacKerell, Bashford, Bellott, Dunbrack, Evanseck, Field, Fischer, Gao, Guo, Ha, et al, J Phys Chem, 102, 3586 (1998).

pair_style lj/class2 command

pair_style lj/class2/cuda command

pair_style lj/class2/gpu command

pair_style lj/class2/coul/cut command

pair_style lj/class2/coul/cut/cuda command

pair_style lj/class2/coul/long command

pair_style lj/class2/coul/long/cuda command

pair_style lj/class2/coul/long/gpu command

Syntax:

```
pair_style style args
```

- style = *lj/class2* or *lj/class2/coul/cut* or *lj/class2/coul/long*
- args = list of arguments for a particular style

```
lj/class2 args = cutoff
    cutoff = global cutoff for class 2 interactions (distance units)
  lj/class2/coul/cut args = cutoff (cutoff2)
    cutoff = global cutoff for class 2 (and Coulombic if only 1 arg) (distance units)
    cutoff2 = global cutoff for Coulombic (optional) (distance units)
  lj/class2/coul/long args = cutoff (cutoff2)
    cutoff = global cutoff for class 2 (and Coulombic if only 1 arg) (distance units)
    cutoff2 = global cutoff for Coulombic (optional) (distance units)
```

Examples:

```
pair_style lj/class2 10.0
pair_coeff * * 100.0 2.5
pair_coeff 1 2* 100.0 2.5 9.0

pair_style lj/class2/coul/cut 10.0
pair_style lj/class2/coul/cut 10.0 8.0
pair_coeff * * 100.0 3.0
pair_coeff 1 1 100.0 3.5 9.0
pair_coeff 1 1 100.0 3.5 9.0 9.0

pair_style lj/class2/coul/long 10.0
pair_style lj/class2/coul/long 10.0 8.0
pair_coeff * * 100.0 3.0
pair_coeff 1 1 100.0 3.5 9.0
```

Description:

The *lj/class2* styles compute a 6/9 Lennard−Jones potential given by

$$E = \epsilon \left[2 \left(\frac{\sigma}{r} \right)^9 - 3 \left(\frac{\sigma}{r} \right)^6 \right] \qquad r < r_c$$

Rc is the cutoff.

The *lj/class2/coul/cut* and *lj/class2/coul/long* styles add a Coulombic term as described for the lj/cut pair styles.

See (Sun) for a description of the COMPASS class2 force field.

The following coefficients must be defined for each pair of atoms types via the pair_coeff command as in the examples above, or in the data file or restart files read by the read_data or read_restart commands, or by mixing as described below:

- epsilon (energy units)
- sigma (distance units)
- cutoff1 (distance units)
- cutoff2 (distance units)

The latter 2 coefficients are optional. If not specified, the global class 2 and Coulombic cutoffs are used. If only one cutoff is specified, it is used as the cutoff for both class 2 and Coulombic interactions for this type pair. If both coefficients are specified, they are used as the class 2 and Coulombic cutoffs for this type pair. You cannot specify 2 cutoffs for style *lj/class2*, since it has no Coulombic terms.

For *lj/class2/coul/long* only the class 2 cutoff can be specified since a Coulombic cutoff cannot be specified for an individual I,J type pair. All type pairs use the same global Coulombic cutoff specified in the pair_style command.

If the pair_coeff command is not used to define coefficients for a particular I != J type pair, the mixing rule for epsilon and sigma for all class2 potentials is to use the *sixthpower* formulas documented by the pair_modify command. The pair_modify mix setting is thus ignored for class2 potentials for epsilon and sigma. However it is still followed for mixing the cutoff distance.

Styles with a *cuda*, *gpu*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in this section of the manual. The accelerated styles take the same arguments and should produce the same results, except for round−off and precision issues.

These accelerated styles are part of the "user−cuda", "gpu", and "opt" packages respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the −suffix command−line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See this section of the manual for more instructions on how to use the accelerated styles effectively.

Mixing, shift, table, tail correction, restart, rRESPA info:

For atom type pairs I,J and I != J, the epsilon and sigma coefficients and cutoff distance for all of the lj/class2 pair styles can be mixed. Epsilon and sigma are always mixed with the value *sixthpower*. The cutoff distance is mixed by whatever option is set by the pair_modify command (default = geometric). See the "pair_modify" command

for details.

All of the lj/class2 pair styles support the pair_modify shift option for the energy of the Lennard−Jones portion of the pair interaction.

The *lj/class2/coul/long* pair style does not support the pair_modify table option since a tabulation capability has not yet been added to this potential.

All of the lj/class2 pair styles support the pair_modify tail option for adding a long−range tail correction to the energy and pressure of the Lennard−Jones portion of the pair interaction.

All of the lj/class2 pair styles write their information to binary restart files, so pair_style and pair_coeff commands do not need to be specified in an input script that reads a restart file.

All of the lj/class2 pair styles can only be used via the *pair* keyword of the run_style respa command. They do not support the *inner*, *middle*, *outer* keywords.

Restrictions:

These styles are part of the "class2" package. They are only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

Related commands:

pair_coeff

Default: none

(Sun) Sun, J Phys Chem B 102, 7338−7364 (1998).

pair_style cg/cmm command

pair_style cg/cmm/cuda command

pair_style cg/cmm/gpu command

pair_style cg/cmm/coul/cut command

pair_style cg/cmm/coul/cut/cuda command

pair_style cg/cmm/coul/debye/cuda command

pair_style cg/cmm/coul/long command

pair_style cg/cmm/coul/long/gpu command

pair_style cg/cmm/coul/long/cuda command

Syntax:

```
pair_style style args
```

- style = *cg/cmm* or *cg/cmm/coul/cut* or *cg/cmm/coul/long*
- args = list of arguments for a particular style

```
cg/cmm args = cutoff
    cutoff = global cutoff for Lennard Jones interactions (distance units)
  cg/cmm/coul/cut args = cutoff (cutoff2) (kappa)
    cutoff = global cutoff for LJ (and Coulombic if only 1 arg) (distance units)
    cutoff2 = global cutoff for Coulombic (optional) (distance units)
    kappa = Debye length (optional, defaults to 0.0 = disabled) (inverse distance units)
  cg/cmm/coul/long args = cutoff (cutoff2)
    cutoff = global cutoff for LJ (and Coulombic if only 1 arg) (distance units)
    cutoff2 = global cutoff for Coulombic (optional) (distance units)
```

Examples:

```
pair_style cg/cmm 2.5
pair_coeff 1 1 lj12_6 1 1.1 2.8

pair_style cg/cmm/coul/cut 10.0 12.0
pair_coeff 1 1 lj9_6  100.0 3.5 9.0
pair_coeff 1 1 lj12_4 100.0 3.5 9.0 9.0

pair_style cg/cmm/coul/long 10.0
pair_style cg/cmm/coul/long 10.0 8.0
pair_coeff 1 1 lj9_6 100.0 3.5 9.0
```

Description:

The *cg/cmm* styles compute a 9/6, 12/4, or 12/6 Lennard−Jones potential, given by

$$E = \frac{27}{4}\epsilon\left[\left(\frac{\sigma}{r}\right)^9 - \left(\frac{\sigma}{r}\right)^6\right] \qquad r < r_c$$
$$E = \frac{3\sqrt{3}}{2}\epsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^4\right] \qquad r < r_c$$
$$E = 4\epsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^6\right] \qquad r < r_c$$

as required for the CMM Coarse−grained MD parametrization discussed in (Shinoda) and (DeVane). Rc is the cutoff.

Style *cg/cmm/coul/cut* adds a Coulombic pairwise interaction given by

$$E = \frac{Cq_iq_j}{\epsilon r} \qquad r < r_c$$

where C is an energy−conversion constant, Qi and Qj are the charges on the 2 atoms, and epsilon is the dielectric constant which can be set by the dielectric command. If one cutoff is specified in the pair_style command, it is used for both the LJ and Coulombic terms. If two cutoffs are specified, they are used as cutoffs for the LJ and Coulombic terms respectively.

This style also contains an additional exp() damping factor to the Coulombic term, given by

$$E = \frac{Cq_iq_j}{\epsilon r}\exp(-\kappa r) \qquad r < r_c$$

where kappa is the Debye length (kappa=0.0 is the unscreened coulomb). This potential is another way to mimic the screening effect of a polar solvent.

Style *cg/cmm/coul/long* computes the same Coulombic interactions as style *cg/cmm/coul/cut* except that an additional damping factor is applied to the Coulombic term so it can be used in conjunction with the kspace_style command and its *ewald* or *pppm* option. The Coulombic cutoff specified for this style means that pairwise interactions within this distance are computed directly; interactions outside that distance are computed in reciprocal space.

The following coefficients must be defined for each pair of atoms types via the pair_coeff command as in the examples above, or in the data file or restart files read by the read_data or read_restart commands, or by mixing as described below:

- cg_type (lj9_6, lj12_4, or lj12_6)
- epsilon (energy units)
- sigma (distance units)
- cutoff1 (distance units)
- cutoff2 (distance units)

Note that sigma is defined in the LJ formula as the zero−crossing distance for the potential, not as the energy minimum. The prefactors are chosen so that the potential minimum is at −epsilon.

The latter 2 coefficients are optional. If not specified, the global LJ and Coulombic cutoffs specified in the pair_style command are used. If only one cutoff is specified, it is used as the cutoff for both LJ and Coulombic

interactions for this type pair. If both coefficients are specified, they are used as the LJ and Coulombic cutoffs for this type pair.

For *cg/cmm/coul/long* only the LJ cutoff can be specified since a Coulombic cutoff cannot be specified for an individual I,J type pair. All type pairs use the same global Coulombic cutoff specified in the pair_style command.

Styles with a *cuda*, *gpu*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in this section of the manual. The accelerated styles take the same arguments and should produce the same results, except for round−off and precision issues.

These accelerated styles are part of the "user−cuda", "gpu", and "opt" packages respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the −suffix command−line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See this section of the manual for more instructions on how to use the accelerated styles effectively.

Mixing, shift, table, tail correction, restart, and rRESPA info:

For atom type pairs I,J and I != J, the epsilon and sigma coefficients and cutoff distance for all of the cg/cmm pair styles *cannot* be mixed, since different pairs may have different exponents. So all parameters for all pairs have to be specified explicitly through the "pair_coeff" command. Defining then in a data file is also not supported, due to limitations of that file format.

All of the cg/cmm pair styles support the pair_modify shift option for the energy of the Lennard−Jones portion of the pair interaction.

The *cg/cmm/coul/long* pair styles support the pair_modify table option since they can tabulate the short−range portion of the long−range Coulombic interaction.

All of the cg/cmm pair styles write their information to binary restart files, so pair_style and pair_coeff commands do not need to be specified in an input script that reads a restart file.

The cg/cmm, cg/cmm/coul/cut and lj/cut/coul/long pair styles support the use of the *inner*, *middle*, and *outer* keywords of the run_style respa command, meaning the pairwise forces can be partitioned by distance at different levels of the rRESPA hierarchy. See the run_style command for details.

Restrictions:

All of the cg/cmm pair styles are part of the "user−cg−cmm" package. The *cg/cmm/coul/long* style also requires the "kspace" package to be built (which is enabled by default). They are only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

Related commands:

pair_coeff, angle_style cg/cmm

Default: none

(Shinoda) Shinoda, DeVane, Klein, Mol Sim, 33, 27 (2007).

(DeVane) Shinoda, DeVane, Klein, Soft Matter, 4, 2453−2462 (2008).

pair_coeff command

Syntax:

```
pair_coeff I J args
```

- I,J = atom types (see asterisk form below)
- args = coefficients for one or more pairs of atom types

Examples:

```
pair_coeff 1 2 1.0 1.0 2.5
pair_coeff 2 * 1.0 1.0
pair_coeff 3* 1*2 1.0 1.0 2.5
pair_coeff * * 1.0 1.0
pair_coeff * * nialhjea 1 1 2
pair_coeff * 3 morse.table ENTRY1
pair_coeff 1 2 lj/cut 1.0 1.0 2.5 (for pair_style hybrid)
```

Description:

Specify the pairwise force field coefficients for one or more pairs of atom types. The number and meaning of the coefficients depends on the pair style. Pair coefficients can also be set in the data file read by the read_data command or in a restart file.

I and J can be specified in one of two ways. Explicit numeric values can be used for each, as in the 1st example above. I <= J is required. LAMMPS sets the coefficients for the symmetric J,I interaction to the same values.

A wildcard asterisk can be used in place of or in conjunction with the I,J arguments to set the coefficients for multiple pairs of atom types. This takes the form "*" or "*n" or "n*" or "m*n". If N = the number of atom types, then an asterisk with no numeric values means all types from 1 to N. A leading asterisk means all types from 1 to n (inclusive). A trailing asterisk means all types from n to N (inclusive). A middle asterisk means all types from m to n (inclusive). Note that only type pairs with I <= J are considered; if asterisks imply type pairs where J < I, they are ignored.

Note that a pair_coeff command can override a previous setting for the same I,J pair. For example, these commands set the coeffs for all I,J pairs, then overwrite the coeffs for just the I,J = 2,3 pair:

```
pair_coeff * * 1.0 1.0 2.5
pair_coeff 2 3 2.0 1.0 1.12
```

A line in a data file that specifies pair coefficients uses the exact same format as the arguments of the pair_coeff command in an input script, with the exception of the I,J type arguments. In each line of the "Pair Coeffs" section of a data file, only a single type I is specified, which sets the coefficients for type I interacting with type I. This is because the section has exactly N lines, where N = the number of atom types. For this reason, the wild−card asterisk should also not be used as part of the I argument. Thus in a data file, the line corresponding to the 1st example above would be listed as

```
2 1.0 1.0 2.5
```

For many potentials, if coefficients for type pairs with I != J are not set explicitly by a pair_coeff command, the values are inferred from the I,I and J,J settings by mixing rules; see the pair_modify command for a discussion.

Details on this option as it pertains to individual potentials are described on the doc page for the potential.

Here is an alphabetic list of pair styles defined in LAMMPS. Click on the style to display the formula it computes, arguments specified in the pair_style command, and coefficients specified by the associated pair_coeff command:

- pair_style hybrid − multiple styles of pairwise interactions
- pair_style hybrid/overlay − multiple styles of superposed pairwise interactions

- pair_style adp − angular dependent potential (ADP) of Mishin
- pair_style airebo − AIREBO potential of Stuart
- pair_style born − Born−Mayer−Huggins potential
- pair_style born/coul/long − Born−Mayer−Huggins with long−range Coulomb
- pair_style buck − Buckingham potential
- pair_style buck/coul/cut − Buckingham with cutoff Coulomb
- pair_style buck/coul/long − Buckingham with long−range Coulomb
- pair_style colloid − integrated colloidal potential
- pair_style coul/cut − cutoff Coulombic potential
- pair_style coul/debye − cutoff Coulombic potential with Debye screening
- pair_style coul/long − long−range Coulombic potential
- pair_style dipole/cut − point dipoles with cutoff
- pair_style dpd − dissipative particle dynamics (DPD)
- pair_style dpd/tstat − DPD thermostatting
- pair_style dsmc − Direct Simulation Monte Carlo (DSMC)
- pair_style eam − embedded atom method (EAM)
- pair_style eam/alloy − alloy EAM
- pair_style eam/fs − Finnis−Sinclair EAM
- pair_style eim − embedded ion method (EIM)
- pair_style gauss − Gaussian potential
- pair_style gayberne − Gay−Berne ellipsoidal potential
- pair_style gran/hertz/history − granular potential with Hertzian interactions
- pair_style gran/hooke − granular potential with history effects
- pair_style gran/hooke/history − granular potential without history effects
- pair_style hbond/dreiding/lj − DREIDING hydrogen bonding LJ potential
- pair_style hbond/dreiding/morse − DREIDING hydrogen bonding Morse potential
- pair_style lj/charmm/coul/charmm − CHARMM potential with cutoff Coulomb
- pair_style lj/charmm/coul/charmm/implicit − CHARMM for implicit solvent
- pair_style lj/charmm/coul/long − CHARMM with long−range Coulomb
- pair_style lj/class2 − COMPASS (class 2) force field with no Coulomb
- pair_style lj/class2/coul/cut − COMPASS with cutoff Coulomb
- pair_style lj/class2/coul/long − COMPASS with long−range Coulomb
- pair_style lj/cut − cutoff Lennard−Jones potential with no Coulomb
- pair_style lj/cut/coul/cut − LJ with cutoff Coulomb
- pair_style lj/cut/coul/debye − LJ with Debye screening added to Coulomb
- pair_style lj/cut/coul/long − LJ with long−range Coulomb
- pair_style lj/cut/coul/long/tip4p − LJ with long−range Coulomb for TIP4P water
- pair_style lj/expand − Lennard−Jones for variable size particles
- pair_style lj/gromacs − GROMACS−style Lennard−Jones potential
- pair_style lj/gromacs/coul/gromacs − GROMACS−style LJ and Coulombic potential
- pair_style lj/smooth − smoothed Lennard−Jones potential
- pair_style lj96/cut − Lennard−Jones 9/6 potential
- pair_style lubricate − hydrodynamic lubrication forces
- pair_style meam − modified embedded atom method (MEAM)

- pair_style morse − Morse potential
- pair_style peri/lps − peridynamic LPS potential
- pair_style peri/pmb − peridynamic PMB potential
- pair_style reax − ReaxFF potential
- pair_style rebo − 2nd−generation REBO potential of Brenner
- pair_style resquared − Everaers RE−Squared ellipsoidal potential
- pair_style soft − Soft (cosine) potential
- pair_style sw − Stillinger−Weber 3−body potential
- pair_style table − tabulated pair potential
- pair_style tersoff − Tersoff 3−body potential
- pair_style tersoff/zbl − Tersoff/ZBL 3−body potential
- pair_style yukawa − Yukawa potential
- pair_style yukawa/colloid − screened Yukawa potential for finite−size particles

There are also additional pair styles submitted by users which are included in the LAMMPS distribution. The list of these with links to the individual styles are given in the pair section of this page.

There are also additional accelerated pair styles included in the LAMMPS distribution for faster performance on CPUs and GPUs. The list of these with links to the individual styles are given in the pair section of this page.

Restrictions:

This command must come after the simulation box is defined by a read_data, read_restart, or create_box command.

Related commands:

pair_style, pair_modify, read_data, read_restart, pair_write

Default: none

pair_style colloid command

Syntax:

```
pair_style colloid cutoff
```

- cutoff = global cutoff for colloidal interactions (distance units)

Examples:

```
pair_style colloid 10.0
pair_coeff * *  25 1.0 10.0 10.0
pair_coeff 1 1 144 1.0 0.0 0.0 3.0
pair_coeff 1 2  75.398 1.0 0.0 10.0 9.0
pair_coeff 2 2  39.478 1.0 10.0 10.0 25.0
```

Description:

Style *colloid* computes pairwise interactions between large colloidal particles and small solvent particles using 3 formulas. A colloidal particle has a size > sigma; a solvent particle is the usual Lennard−Jones particle of size sigma.

The colloid−colloid interaction energy is given by

$$
\begin{aligned}
U_A &= -\frac{A_{cc}}{6}\left[\frac{2a_1a_2}{r^2-(a_1+a_2)^2}+\frac{2a_1a_2}{r^2-(a_1-a_2)^2}+\ln\left(\frac{r^2-(a_1+a_2)^2}{r^2-(a_1-a_2)^2}\right)\right] \\
U_R &= \frac{A_{cc}}{37800}\frac{\sigma^6}{r}\Bigg[\\
&\quad \frac{r^2-7r\,(a_1+a_2)+6\,(a_1^2+7a_1a_2+a_2^2)}{(r-a_1-a_2)^7} \\
&\quad +\frac{r^2+7r\,(a_1+a_2)+6\,(a_1^2+7a_1a_2+a_2^2)}{(r+a_1+a_2)^7} \\
&\quad -\frac{r^2+7r\,(a_1-a_2)+6\,(a_1^2-7a_1a_2+a_2^2)}{(r+a_1-a_2)^7} \\
&\quad -\frac{r^2-7r\,(a_1-a_2)+6\,(a_1^2-7a_1a_2+a_2^2)}{(r-a_1+a_2)^7}\Bigg] \\
U &= U_A+U_R, \qquad r<r_c
\end{aligned}
$$

where A_cc is the Hamaker constant, a1 and a2 are the radii of the two colloidal particles, and Rc is the cutoff. This equation results from describing each colloidal particle as an integrated collection of Lennard−Jones particles of size sigma and is derived in (Everaers).

The colloid−solvent interaction energy is given by

$$U = \frac{2\,a^3\,\sigma^3\,A_{cs}}{9\,(a^2-r^2)^3}\left[1-\frac{(5\,a^6+45\,a^4\,r^2+63\,a^2\,r^4+15\,r^6)\,\sigma^6}{15\,(a-r)^6\,(a+r)^6}\right],\quad r<r_c$$

where A_cs is the Hamaker constant, a is the radius of the colloidal particle, and Rc is the cutoff. This formula is derived from the colloid−colloid interaction, letting one of the particle sizes go to zero.

The solvent−solvent interaction energy is given by the usual Lennard−Jones formula

$$U = \frac{A_{ss}}{36}\left[\left(\frac{\sigma}{r}\right)^{12}-\left(\frac{\sigma}{r}\right)^{6}\right],\quad r<r_c$$

with A_ss set appropriately, which results from letting both particle sizes go to zero.

When used in combination with pair_style yukawa/colloid, the two terms become the so−called DLVO potential, which combines electrostatic repulsion and van der Waals attraction.

The following coefficients must be defined for each pair of atoms types via the pair_coeff command as in the examples above, or in the data file or restart files read by the read_data or read_restart commands, or by mixing as described below:

- A (energy units)
- sigma (distance units)
- d1 (distance units)
- d2 (distance units)
- cutoff (distance units)

A is the Hamaker energy prefactor and should typically be set as follows:

- A_cc = colloid/colloid = 4 pi^2 = 39.5
- A_cs = colloid/solvent = sqrt(A_cc*A_ss)
- A_ss = solvent/solvent = 144 (assuming epsilon = 1, so that 144/36 = 4)

Sigma is the size of the solvent particle or the constituent particles integrated over in the colloidal particle and should typically be set as follows:

- Sigma_cc = colloid/colloid = 1.0
- Sigma_cs = colloid/solvent = arithmetic mixing between colloid sigma and solvent sigma
- Sigma_ss = solvent/solvent = 1.0 or whatever size the solvent particle is

Thus typically Sigma_cs = 1.0, unless the solvent particle's size != 1.0.

D1 and d2 are particle diameters, so that d1 = 2*a1 and d2 = 2*a2 in the formulas above. Both d1 and d2 must be values >= 0. If d1 > 0 and d2 > 0, then the pair interacts via the colloid−colloid formula above. If d1 = 0 and d2 = 0, then the pair interacts via the solvent−solvent formula. I.e. a d value of 0 is a Lennard−Jones particle of size sigma. If either d1 = 0 or d2 = 0 and the other is larger, then the pair interacts via the colloid−solvent formula.

Note that the diameter of a particular particle type may appear in multiple pair_coeff commands, as it interacts with other particle types. You should insure the particle diameter is specified consistently each time it appears.

The last coefficient is optional. If not specified, the global cutoff specified in the pair_style command is used. However, you typically want different cutoffs for interactions between different particle sizes. E.g. if colloidal particles of diameter 10 are used with solvent particles of diameter 1, then a solvent−solvent cutoff of 2.5 would correspond to a colloid−colloid cutoff of 25. A good rule−of−thumb is to use a colloid−solvent cutoff that is half the big diameter + 4 times the small diameter. I.e. 9 = 5 + 4 for the colloid−solvent cutoff in this case.

IMPORTANT NOTE: When using pair_style colloid for a mixture with 2 (or more) widely different particles sizes (e.g. sigma=10 colloids in a background sigam=1 LJ fluid), you will likely want to use these commands for efficiency: neighbor multi and communicate multi.

Mixing, shift, table, tail correction, restart, rRESPA info:

For atom type pairs I,J and I != J, the A, sigma, d1, and d2 coefficients and cutoff distance for this pair style can be mixed. A is an energy value mixed like a LJ epsilon. D1 and d2 are distance values and are mixed like sigma. The default mix value is *geometric*. See the "pair_modify" command for details.

This pair style supports the pair_modify shift option for the energy of the pair interaction.

The pair_modify table option is not relevant for this pair style.

This pair style does not support the pair_modify tail option for adding long−range tail corrections to energy and pressure.

This pair style writes its information to binary restart files, so pair_style and pair_coeff commands do not need to be specified in an input script that reads a restart file.

This pair style can only be used via the *pair* keyword of the run_style respa command. It does not support the *inner*, *middle*, *outer* keywords.

Restrictions:

This style is part of the "colloid" package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

Normally, this pair style should be used with finite−size particles which have a diameter, e.g. see the atom_style sphere command. However, this is not a requirement, since the only definition of particle size is via the pair_coeff parameters for each type. In other words, the physical radius of the particle is ignored. Thus you should insure that the d1,d2 parameters you specify are consistent with the physical size of the particles of that type.

Per−particle polydispersity is not yet supported by this pair style; only per−type polydispersity is enabled via the pair_coeff parameters.

Related commands:

pair_coeff

Default: none

(Everaers) Everaers, Ejtehadi, Phys Rev E, 67, 041710 (2003).

pair_style comb command

Syntax:

```
pair_style comb
```

Examples:

```
pair_style comb
pair_coeff * * ../potentials/ffield.comb Si
pair_coeff * * ../potentials/ffield.comb Hf Si O
```

Description:

Style *comb* computes a variable charge COMB (Charge−Optimized Many−Body) potential as described in (COMB_1) and (COMB_2). The energy E of a system of atoms is given by

$$\begin{aligned} E_T &= \sum_i [E_i^S + \frac{1}{2}\sum_{j\neq i} V_{ij}(r_{ij},q_i,q_j) + E_i^{BB}] \\ V_{ij}(r_{ij},q_i,q_j) &= U_{ij}^R(r_{ij}) + U_{ij}^A(r_{ij},q_i,q_j) + U_{ij}^I(r_{ij},q_i,q_j) + U_{ij}^V(r_{ij}) \end{aligned}$$

where E_T is the total potential energy of the system, $E^S{}_i$ is the self−energy term of atom i, V_{ij} is the interatomic potential between the ith and jth atoms, r_{ij} is the distance of the atoms i and j, and q_i and q_j are charges of the atoms, and $E^{BB}{}_i$ is the bond−bending term of atom i.

The interatomic potential energy V_{ij} consists of four components: two−body short−range repulsion, $U^R{}_{ij}$, many−body short−range attraction, $U^A{}_{ij}$, long−range Coulombic electrostatic interaction, $U^I{}_{ij}$, and van der Waals energy, $U^V{}_{ij}$, which are defined as:

$$\begin{aligned} U_{ij}^R(r_{ij}) &= f_{S_{ij}} A_{ij} \exp(-\lambda_{ij} r_{ij}) \\ U_{ij}^A(r_{ij},q_i,q_j) &= -f_{S_{ij}} b_{ij} B_{ij} \exp(-\alpha_{ij} r_{ij}) \\ U_{ij}^I(r_{ij},q_i,q_j) &= J_{ij}(r_{ij}) q_i q_j \\ U_{ij}^V(r_{ij}) &= f_{L_{ij}} (C_{VDW_i} C_{VDW_j})^{\frac{1}{2}} / r_{ij}^6 \end{aligned}$$

The short−range repulsion and attraction are based on the Tersoff potential (see the pair_style tersoff command); thus for a zero−charge pure element system with no van der Waals interaction, the COMB potential reduces to Tersoff potential, typically truncated at a short cutoff, e.g. 3 to 4 Angstroms. The long−range Coulombic term uses the Wolf summation method described in Wolf, spherically truncated at a longer cutoff, e.g. 12 Angstroms.

The COMB potential is a variable charge potential. The equilibrium charge on each atom is calculated by the electronegativity equalization (QEq) method. See Rick for further details. This is implemented by the fix qeq/comb command, which should normally be specified in the input script when running a model with the COMB potential. The fix qeq/comb command has options that determine how often charge equilibration is performed, its convergence criterion, and which atoms are included in the calculation.

Only a single pair_coeff command is used with the *comb* style which specifies the COMB potential file with parameters for all needed elements. These are mapped to LAMMPS atom types by specifying N additional arguments after the potential file in the pair_coeff command, where N is the number of LAMMPS atom types. The provided potential file *ffield.comb* contains all currently−available COMB parameterizations: for Si, Cu, Hf, Ti, O, their oxides and Zr, Zn and U metals.

For example, if your LAMMPS simulation of a $Si/SiO_2/ HfO_2$ interface has 4 atom types, and you want the 1st and last to be Si, the 2nd to be Hf, and the 3rd to be O, and you would use the following pair_coeff command:

```
pair_coeff * * ../potentials/ffield.comb Si Hf O Si
```

The first two arguments must be * * so as to span all LAMMPS atom types. The first and last Si arguments map LAMMPS atom types 1 and 4 to the Si element in the *ffield.comb* file. The second Hf argument maps LAMMPS atom type 2 to the Hf element, and the third O argument maps LAMMPS atom type 3 to the O element in the potential file. If a mapping value is specified as NULL, the mapping is not performed. This can be used when a *comb* potential is used as part of the *hybrid* pair style. The NULL values are placeholders for atom types that will be used with other potentials.

The *ffield.comb* potential file is in the *potentials* directory of the LAMMPS distribution. Lines that are not blank or comments (starting with #) define parameters for a triplet of elements. The 49 parameters in a single entry correspond to coefficients in the formula above:

- element 1 (the center atom in a 3−body interaction)
- element 2 (the atom bonded to the center atom)
- element 3 (the atom influencing the 1−2 bond in a bond−order sense)
- m
- c
- d
- h (cos_theta0 (can be a value −1 or 1))
- n
- beta
- lambda21, lambda2 of element 1 (1/distance units)
- lambda22, lambda2 of element 2 (1/distance units)
- B of element 1 (energy units)
- B of element 2 (energy units)
- R (cutoff, distance units, 0.5*(r_outer + r_inner))
- D (cutoff, distance units, R − r_inner)
- lambda11, lambda1 of element 1 (1/distance units)
- lambda12, lambda1 of element 2 (1/distance units)
- A of element 1 (energy units)
- A of element 2 (energy units)
- K_LP_1 (energy units, 1st order Legendre polynomial coefficient)
- K_LP_3 (energy units, 3rd order Legendre polynomial coefficient)
- K_LP_6 (energy units, 6th order Legendre polynomial coefficient)
- A123 (cos_theta, theta = equilibrium MOM or OMO bond angles)
- Aconf (cos_theta, theta = equilibrium MOM or OMO bond−bending coefficient)
- addrep (energy units, additional repulsion)
- R_omiga_a (unit−less scaler for A)
- R_omiga_b (unit−less scaler for B)
- R_omiga_c (unit−less scaler for 0.5*(lambda21+lambda22))
- R_omiga_d (unit−less scaler for 0.5*(lambda11+lambda12))
- QL1 (charge units, lower charge limit for element 1)
- QU1 (charge units, upper charge limit for element 1)

- DL1 (distance units, ion radius of element 1 with charge QL1)
- DU1 (distance units, ion radius of element 1 with charge QU1)
- QL2 (charge units, lower charge limit for element 2)
- QU2 (charge units, upper charge limit for element 2)
- DL2 (distance units, ion radius of element 2 with charge QL2)
- DU2 (distance units, ion radius of element 2 with charge QU2)
- chi (energy units, self energy 1st power term)
- dJ (energy units, self energy 2nd power term)
- dK (energy units, self energy 3rd power term)
- dL (energy units, self energy 4th power term)
- dM (energy units, self energy 6th power term)
- esm (distance units, orbital exponent)
- cmn1 (self energy penalty, rho 1 of element 1)
- cml1 (self energy penalty, rho 1 of element 2)
- cmn2 (self energy penalty, rho 2 of element 1)
- cmn2 (self energy penalty, rho 2 of element 2)
- coulcut (long range Coulombic cutoff, distance units)
- hfocor (coordination term)

The parameterization of COMB potentials start with a pure element (e.g. Si, Cu) then extend to its oxide and polymorphs (e.g. SiO_2, Cu_2O). For interactions not involving oxygen (e.g. Si−Cu or Hf−Zr), the COMB potential uses a mixing rule to generate these parameters. For furthur details on the parameterization and parameters, see the Tersoff doc page and the COMB publications (COMB_1) and (COMB_2). For more details on 3−body interaction types (e.g. SiSiO vs SiOSi), the mixing rule, and how to generate the potential file, please see the Tersoff doc page.

In the potentials directory, the file *ffield.comb* provides the LAMMPS parameters for COMB's Si, Cu, Ti, Hf and their oxides, as well as pure U, Zn and Zr metals. This file can be used for pure elements (e.g. Si, Zr), binary oxides, binary alloys (e.g. SiCu, TiZr), and complex systems. Note that alloys and complex systems require all 3−body entries be pre−defined in the potential file.

Mixing, shift, table, tail correction, restart, rRESPA info:

For atom type pairs I,J and I != J, where types I and J correspond to two different element types, mixing is performed by LAMMPS as described above from values in the potential file.

This pair style does not support the pair_modify shift, table, and tail options.

This pair style does not write its information to binary restart files, since it is stored in potential files. Thus, you need to re−specify the pair_style, pair_coeff, and fix qeq/comb commands in an input script that reads a restart file.

This pair style can only be used via the *pair* keyword of the run_style respa command. It does not support the *inner*, *middle*, *outer* keywords.

Restrictions:

This pair style is part of the "manybody" package. It is only enabled if LAMMPS was built with that package (which it is by default). See the Making LAMMPS section for more info.

This pair style requires the newton setting to be "on" for pair interactions.

The COMB potentials in the *ffield.comb* file provided with LAMMPS (see the potentials directory) are parameterized for metal units. You can use the COMB potential with any LAMMPS units, but you would need to create your own COMB potential file with coefficients listed in the appropriate units if your simulation doesn't use "metal" units.

Related commands:

pair_style, pair_coeff, fix_qeq/comb

Default: none

(COMB_1) J. Yu, S. B. Sinnott, S. R. Phillpot, Phys Rev B, 75, 085311 (2007),

(COMB_2) T.−R. Shan, B. D. Devine, T. W. Kemper, S. B. Sinnott, S. R. Phillpot, Phys Rev B, 81, 125328 (2010).

(Tersoff) J. Tersoff, Phys Rev B, 37, 6991 (1988).

(Rick) S. W. Rick, S. J. Stuart, B. J. Berne, J Chem Phys 101, 16141 (1994).

(Wolf) D. Wolf, P. Keblinski, S. R. Phillpot, J. Eggebrecht, J Chem Phys, 110, 8254 (1999).

pair_style coul/cut command

pair_style coul/debye command

pair_style coul/long command

pair_style coul/long/gpu command

Syntax:

```
pair_style coul/cut cutoff
pair_style coul/debye kappa cutoff
pair_style coul/long cutoff
pair_style coul/long/gpu cutoff
```

- cutoff = global cutoff for Coulombic interactions
- kappa = Debye length (inverse distance units)

Examples:

```
pair_style coul/cut 2.5
pair_coeff * *
pair_coeff 2 2 3.5

pair_style coul/debye 1.4 3.0
pair_coeff * *
pair_coeff 2 2 3.5

pair_style coul/long 10.0
pair_coeff * *
```

Description:

The *coul/cut* style computes the standard Coulombic interaction potential given by

$$E = \frac{Cq_iq_j}{\epsilon r} \qquad r < r_c$$

where C is an energy−conversion constant, Qi and Qj are the charges on the 2 atoms, and epsilon is the dielectric constant which can be set by the dielectric command. The cutoff Rc truncates the interaction distance.

Style *coul/debye* adds an additional exp() damping factor to the Coulombic term, given by

$$E = \frac{Cq_iq_j}{\epsilon r} \exp(-\kappa r) \qquad r < r_c$$

where kappa is the Debye length. This potential is another way to mimic the screening effect of a polar solvent.

Style *coul/long* computes the same Coulombic interactions as style *coul/cut* except that an additional damping factor is applied so it can be used in conjunction with the kspace_style command and its *ewald* or *pppm* option. The Coulombic cutoff specified for this style means that pairwise interactions within this distance are computed directly; interactions outside that distance are computed in reciprocal space.

These potentials are designed to be combined with other pair potentials via the pair_style hybrid/overlay command. This is because they have no repulsive core. Hence if they are used by themselves, there will be no repulsion to keep two oppositely charged particles from overlapping each other.

The following coefficients must be defined for each pair of atoms types via the pair_coeff command as in the examples above, or in the data file or restart files read by the read_data or read_restart commands, or by mixing as described below:

- cutoff (distance units)

For *coul/cut* and *coul/debye*, the cutoff coefficient is optional. If it is not used (as in some of the examples above), the default global value specified in the pair_style command is used.

For *coul/long* no cutoff can be specified for an individual I,J type pair via the pair_coeff command. All type pairs use the same global Coulombic cutoff specified in the pair_style command.

Styles with a *cuda*, *gpu*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in this section of the manual. The accelerated styles take the same arguments and should produce the same results, except for round−off and precision issues.

These accelerated styles are part of the "user−cuda", "gpu", and "opt" packages respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the −suffix command−line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See this section of the manual for more instructions on how to use the accelerated styles effectively.

Mixing, shift, table, tail correction, restart, rRESPA info:

For atom type pairs I,J and I != J, the cutoff distance for the *coul/cut* style can be mixed. The default mix value is *geometric*. See the "pair_modify" command for details.

The pair_modify shift option is not relevant for these pair styles.

The *coul/long* style supports the pair_modify table option for tabulation of the short−range portion of the long−range Coulombic interaction.

These pair styles do not support the pair_modify tail option for adding long−range tail corrections to energy and pressure.

These pair styles write their information to binary restart files, so pair_style and pair_coeff commands do not need to be specified in an input script that reads a restart file.

This pair style can only be used via the *pair* keyword of the run_style respa command. It does not support the *inner*, *middle*, *outer* keywords.

Restrictions:

The *coul/long* style is part of the "kspace" package. It is only enabled if LAMMPS was built with that package (which it is by default). See the Making LAMMPS section for more info.

Related commands:

pair_coeff, pair_style hybrid/overlay

Default: none

pair_style dipole/cut command

pair_style dipole/sf command

Syntax:

```
pair_style dipole/cut cutoff (cutoff2)

pair_style dipole/sf cutoff (cutoff2)
```

- cutoff = global cutoff LJ (and Coulombic if only 1 arg) (distance units)
- cutoff2 = global cutoff for Coulombic (optional) (distance units)

Examples:

```
pair_style dipole/cut 10.0
pair_coeff * * 1.0 1.0
pair_coeff 2 3 1.0 1.0 2.5 4.0

pair_style dipole/sf 9.0
pair_coeff * * 1.0 1.0
pair_coeff 2 3 1.0 1.0 2.5 4.0
```

Description:

Style *dipole/cut* computes interactions between pairs of particles that each have a charge and/or a point dipole moment. In addition to the usual Lennard−Jones interaction between the particles (Elj) the charge−charge (Eqq), charge−dipole (Eqp), and dipole−dipole (Epp) interactions are computed by these formulas for the energy (E), force (F), and torque (T) between particles I and J.

$$
\begin{aligned}
E_{qq} &= \frac{q_i q_j}{r} \\
E_{qp} &= \frac{q}{r^3}(p \bullet \vec{r}) \\
E_{pp} &= \frac{1}{r^3}(\vec{p_i} \bullet \vec{p_j}) - \frac{3}{r^5}(\vec{p_i} \bullet \vec{r})(\vec{p_j} \bullet \vec{r})
\end{aligned}
$$

$$
\begin{aligned}
F_{qq} &= \frac{q_i q_j}{r^3}\vec{r} \\
F_{qp} &= -\frac{q}{r^3}\vec{p} + \frac{3q}{r^5}(\vec{p} \bullet \vec{r})\vec{r} \\
F_{pp} &= \frac{3}{r^5}(\vec{p_i} \bullet \vec{p_j})\vec{r} - \frac{15}{r^7}(\vec{p_i} \bullet \vec{r})(\vec{p_j} \bullet \vec{r})\vec{r} + \frac{3}{r^5}\left[(\vec{p_j} \bullet \vec{r})\vec{p_i} + (\vec{p_i} \bullet \vec{r})\vec{p_j}\right]
\end{aligned}
$$

$$
\begin{aligned}
T_{pq} = T_{ij} &= \frac{q_j}{r^3}(\vec{p_i} \times \vec{r}) \\
T_{qp} = T_{ji} &= -\frac{q_i}{r^3}(\vec{p_j} \times \vec{r}) \\
T_{pp} = T_{ij} &= -\frac{1}{r^3}(\vec{p_i} \times \vec{p_j}) + \frac{3}{r^5}(\vec{p_j} \bullet \vec{r})(\vec{p_i} \times \vec{r}) \\
T_{pp} = T_{ji} &= -\frac{1}{r^3}(\vec{p_j} \times \vec{p_i}) + \frac{3}{r^5}(\vec{p_i} \bullet \vec{r})(\vec{p_j} \times \vec{r})
\end{aligned}
$$

where qi and qj are the charges on the two particles, pi and pj are the dipole moment vectors of the two particles, r is their separation distance, and the vector r = Ri − Rj is the separation vector between the two particles. Note that Eqq and Fqq are simply Coulombic energy and force, Fij = −Fji as symmetric forces, and Tij != −Tji since the torques do not act symmetrically. These formulas are discussed in (Allen) and in (Toukmaji).

Style *dipole/sf* computes "shifted−force" interactions between pairs of particles that each have a charge and/or a point dipole moment. In general, a shifted−force potential is a (sligthly) modified potential containing extra terms that make both the energy and its derivative go to zero at the cutoff distance; this removes (cutoff−related) problems in energy conservation and any numerical instability in the equations of motion (Allen). Shifted−force interactions for the Lennard−Jones (E_LJ), charge−charge (Eqq), charge−dipole (Eqp), dipole−charge (Epq) and dipole−dipole (Epp) potentials are computed by these formulas for the energy (E), force (F), and torque (T) between particles I and J:

$$
\begin{aligned}
E_{LJ} &= 4\epsilon\left\{\left[\left(\frac{\sigma}{r}\right)^{12}-\left(\frac{\sigma}{r}\right)^{6}\right]+\left[6\left(\frac{\sigma}{r_c}\right)^{12}-3\left(\frac{\sigma}{r_c}\right)^{6}\right]\left(\frac{r}{r_c}\right)^{2}-7\left(\frac{\sigma}{r_c}\right)^{12}+4\left(\frac{\sigma}{r_c}\right)^{6}\right\}\\
E_{qq} &= \frac{q_i q_j}{r}\left(1-\frac{r}{r_c}\right)^{2}\\
E_{pq} &= E_{ji}=-\frac{q}{r^3}\left[1-3\left(\frac{r}{r_c}\right)^{2}+2\left(\frac{r}{r_c}\right)^{3}\right](\vec{p}\bullet\vec{r})\\
E_{qp} &= E_{ij}=\frac{q}{r^3}\left[1-3\left(\frac{r}{r_c}\right)^{2}+2\left(\frac{r}{r_c}\right)^{3}\right](\vec{p}\bullet\vec{r})\\
E_{pp} &= \left[1-4\left(\frac{r}{r_c}\right)^{3}+3\left(\frac{r}{r_c}\right)^{4}\right]\left[\frac{1}{r^3}(\vec{p_i}\bullet\vec{p_j})-\frac{3}{r^5}(\vec{p_i}\bullet\vec{r})(\vec{p_j}\bullet\vec{r})\right]
\end{aligned}
$$

$$
\begin{aligned}
F_{LJ} &= \left\{\left[48\epsilon\left(\frac{\sigma}{r}\right)^{12}-24\epsilon\left(\frac{\sigma}{r}\right)^{6}\right]\frac{1}{r^2}-\left[48\epsilon\left(\frac{\sigma}{r_c}\right)^{12}-24\epsilon\left(\frac{\sigma}{r_c}\right)^{6}\right]\frac{1}{r_c^2}\right\}\vec{r}\\
F_{qq} &= \frac{q_i q_j}{r}\left(\frac{1}{r^2}-\frac{1}{r_c^2}\right)\vec{r}\\
F_{pq} &= F_{ij}=-\frac{3q}{r^5}\left[1-\left(\frac{r}{r_c}\right)^{2}\right](\vec{p}\bullet\vec{r})\vec{r}+\frac{q}{r^3}\left[1-3\left(\frac{r}{r_c}\right)^{2}+2\left(\frac{r}{r_c}\right)^{3}\right]\vec{p}\\
F_{qp} &= F_{ij}=\frac{3q}{r^5}\left[1-\left(\frac{r}{r_c}\right)^{2}\right](\vec{p}\bullet\vec{r})\vec{r}-\frac{q}{r^3}\left[1-3\left(\frac{r}{r_c}\right)^{2}+2\left(\frac{r}{r_c}\right)^{3}\right]\vec{p}\\
F_{pp} &= \frac{3}{r^5}\left\{\left[1-\left(\frac{r}{r_c}\right)^{4}\right]\left[(\vec{p_i}\bullet\vec{p_j})-\frac{3}{r^2}(\vec{p_i}\bullet\vec{r})(\vec{p_j}\bullet\vec{r})\right]\vec{r}+\right.\\
&\quad\left.\left[1-4\left(\frac{r}{r_c}\right)^{3}+3\left(\frac{r}{r_c}\right)^{4}\right]\left[(\vec{p_j}\bullet\vec{r})\vec{p_i}+(\vec{p_i}\bullet\vec{r})\vec{p_j}-\frac{2}{r^2}(\vec{p_i}\bullet\vec{r})(\vec{p_j}\bullet\vec{r})\vec{r}\right]\right\}
\end{aligned}
$$

$$
\begin{aligned}
T_{pq}=T_{ij} &= \frac{q_j}{r^3}\left[1-3\left(\frac{r}{r_c}\right)^{2}+2\left(\frac{r}{r_c}\right)^{3}\right](\vec{p_i}\times\vec{r})\\
T_{qp}=T_{ji} &= -\frac{q_i}{r^3}\left[1-3\left(\frac{r}{r_c}\right)^{2}+2\left(\frac{r}{r_c}\right)^{3}\right](\vec{p_j}\times\vec{r})\\
T_{pp}=T_{ij} &= -\frac{1}{r^3}\left[1-4\left(\frac{r}{r_c}\right)^{3}+e3\left(\frac{r}{r_c}\right)^{4}\right](\vec{p_i}\times\vec{p_j})+\\
&\quad\frac{3}{r^5}\left[1-4\left(\frac{r}{r_c}\right)^{3}+3\left(\frac{r}{r_c}\right)^{4}\right](\vec{p_j}\bullet\vec{r})(\vec{p_i}\times\vec{r})\\
T_{pp}=T_{ji} &= -\frac{1}{r^3}\left[1-4\left(\frac{r}{r_c}\right)^{3}+3\left(\frac{r}{r_c}\right)^{4}\right](\vec{p_j}\times\vec{p_i})+\\
&\quad\frac{3}{r^5}\left[1-4\left(\frac{r}{r_c}\right)^{3}+3\left(\frac{r}{r_c}\right)^{4}\right](\vec{p_i}\bullet\vec{r})(\vec{p_j}\times\vec{r})
\end{aligned}
$$

where epsilon and sigma are the standard LJ parameters, r_c is the cutoff, qi and qj are the charges on the two particles, pi and pj are the dipole moment vectors of the two particles, r is their separation distance, and the vector r = Ri − Rj is the separation vector between the two particles. Note that Eqq and Fqq are simply Coulombic energy and force, Fij = −Fji as symmetric forces, and Tij != −Tji since the torques do not act symmetrically. The shifted−force formula for the Lennard−Jones potential is reported in (Stoddard). The original (unshifted) formulas for the electrostatic potentials, forces and torques can be found in (Price). The shifted−force electrostatic potentials have been obtained by applying equation 5.13 of (Allen). The formulas for the corresponding forces and torques have been obtained by applying the 'chain rule' as in appendix C.3 of (Allen).

If one cutoff is specified in the pair_style command, it is used for both the LJ and Coulombic (q,p) terms. If two cutoffs are specified, they are used as cutoffs for the LJ and Coulombic (q,p) terms respectively.

Atoms with dipole moments should be integrated using the fix nve/sphere update dipole command to rotate the dipole moments. The compute temp/sphere command can be used to monitor the temperature, since it includes rotational degrees of freedom. The atom_style dipole command should be used since it defines the point dipoles and their rotational state. The magnitude of the dipole moment for each type of particle can be defined by the dipole command or in the "Dipoles" section of the data file read in by the read_data command. Their initial orientation can be defined by the set dipole command or in the "Atoms" section of the data file.

The following coefficients must be defined for each pair of atoms types via the pair_coeff command as in the examples above, or in the data file or restart files read by the read_data or read_restart commands, or by mixing as described below:

- epsilon (energy units)
- sigma (distance units)
- cutoff1 (distance units)
- cutoff2 (distance units)

The latter 2 coefficients are optional. If not specified, the global LJ and Coulombic cutoffs specified in the pair_style command are used. If only one cutoff is specified, it is used as the cutoff for both LJ and Coulombic interactions for this type pair. If both coefficients are specified, they are used as the LJ and Coulombic cutoffs for this type pair.

Mixing, shift, table, tail correction, restart, rRESPA info:

For atom type pairs I,J and I != J, the epsilon and sigma coefficients and cutoff distances for this pair style can be mixed. The default mix value is *geometric*. See the "pair_modify" command for details.

For atom type pairs I,J and I != J, the A, sigma, d1, and d2 coefficients and cutoff distance for this pair style can be mixed. A is an energy value mixed like a LJ epsilon. D1 and d2 are distance values and are mixed like sigma. The default mix value is *geometric*. See the "pair_modify" command for details.

This pair style does not support the pair_modify shift option for the energy of the Lennard−Jones portion of the pair interaction; such energy goes to zero at the cutoff by construction.

The pair_modify table option is not relevant for this pair style.

This pair style does not support the pair_modify tail option for adding long−range tail corrections to energy and pressure.

This pair style writes its information to binary restart files, so pair_style and pair_coeff commands do not need to be specified in an input script that reads a restart file.

This pair style can only be used via the *pair* keyword of the run_style respa command. It does not support the *inner*, *middle*, *outer* keywords.

Restrictions:

The *dipole/cut* style is part of the "dipole" package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

The *dipole/sf* style is part of the "user−misc" package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

Related commands:

pair_coeff

Default: none

(Allen) Allen and Tildesley, Computer Simulation of Liquids, Clarendon Press, Oxford, 1987.

(Toukmaji) Toukmaji, Sagui, Board, and Darden, J Chem Phys, 113, 10913 (2000).

(Stoddard) Stoddard and Ford, Phys Rev A, 8, 1504 (1973).

(Price) Price, Stone and Alderton, Mol Phys, 52, 987 (1984).

pair_style dpd command

pair_style dpd/tstat command

Syntax:

```
pair_style dpd T cutoff seed
pair_style dpd/tstat Tstart Tstop cutoff seed
```

- T = temperature (temperature units)
- Tstart,Tstop = desired temperature at start/end of run (temperature units)
- cutoff = global cutoff for DPD interactions (distance units)
- seed = random # seed (positive integer)

Examples:

```
pair_style dpd 1.0 2.5 34387
pair_coeff * * 3.0 1.0
pair_coeff 1 1 3.0 1.0 1.0

pair_style dpd/tstat 1.0 1.0 2.5 34387
pair_coeff * * 1.0
pair_coeff 1 1 1.0 1.0
```

Description:

Style *dpd* computes a force field for dissipative particle dynamics (DPD) following the exposition in (Groot).

Style *dpd/tstat* invokes a DPD thermostat on pairwise interactions, which is equivalent to the non−conservative portion of the DPD force field. This thermostat can be used in conjunction with any pair style, and in leiu of per−particle thermostats like fix langevin or ensemble thermostats like Nose Hoover as implemented by fix nvt. To use *dpd/stat* with another pair style, use the pair_style hybrid/overlay command to compute both the desired pair interaction and the thermostat for each pair of particles.

For style *dpd*, the force on atom I due to atom J is given as a sum of 3 terms

$$
\begin{aligned}
\vec{f} &= (F^C + F^D + F^R)\hat{r_{ij}} \qquad r < r_c \\
F^C &= Aw(r) \\
F^D &= -\gamma w^2(r)(\hat{r_{ij}} \bullet \vec{v_{ij}}) \\
F^R &= \sigma w(r)\alpha(\Delta t)^{-1/2} \\
w(r) &= 1 - r/r_c
\end{aligned}
$$

where Fc is a conservative force, Fd is a dissipative force, and Fr is a random force. Rij is a unit vector in the direction Ri − Rj, Vij is the vector difference in velocities of the two atoms = Vi − Vj, alpha is a Gaussian random number with zero mean and unit variance, dt is the timestep size, and w(r) is a weighting factor that varies between 0 and 1. Rc is the cutoff. Sigma is set equal to sqrt(2 Kb T gamma), where Kb is the Boltzmann constant and T is the temperature parameter in the pair_style command.

For style *dpd/tstat*, the force on atom I due to atom J is the same as the above equation, except that the conservative Fc term is dropped. Also, during the run, T is set each timestep to a ramped value from Tstart to Tstop.

For style *dpd*, the pairwise energy associated with style *dpd* is only due to the conservative force term Fc, and is shifted to be zero at the cutoff distance Rc. The pairwise virial is calculated using all 3 terms. For style *dpd/tstat* there is no pairwise energy, but the last two terms of the formula make a contribution to the virial.

For style *dpd*, the following coefficients must be defined for each pair of atoms types via the pair_coeff command as in the examples above, or in the data file or restart files read by the read_data or read_restart commands:

- A (force units)
- gamma (force/velocity units)
- cutoff (distance units)

The last coefficient is optional. If not specified, the global DPD cutoff is used. Note that sigma is set equal to sqrt(2 T gamma), where T is the temperature set by the pair_style command so it does not need to be specified.

For style *dpd/tstat*, the coefficiencts defined for each pair of atoms types via the pair_coeff command is the same, except that A is not included.

Mixing, shift, table, tail correction, restart, rRESPA info:

These pair styles do not support mixing. Thus, coefficients for all I,J pairs must be specified explicitly.

These pair styles do not support the pair_modify shift option for the energy of the pair interaction. Note that as discussed above, the energy due to the conservative Fc term is already shifted to be 0.0 at the cutoff distance Rc.

The pair_modify table option is not relevant for these pair styles.

These pair style do not support the pair_modify tail option for adding long−range tail corrections to energy and pressure.

These pair styles writes their information to binary restart files, so pair_style and pair_coeff commands do not need to be specified in an input script that reads a restart file. Note that the user−specified random number seed is stored in the restart file, so when a simulation is restarted, each processor will re−initialize its random number generator the same way it did initially. This means the random forces will be random, but will not be the same as they would have been if the original simulation had continued past the restart time.

These pair styles can only be used via the *pair* keyword of the run_style respa command. They do not support the *inner*, *middle*, *outer* keywords.

The *dpd/tstat* style can ramp its target temperature over multiple runs, using the *start* and *stop* keywords of the run command. See the run command for details of how to do this.

Restrictions:

The default frequency for rebuilding neighbor lists is every 10 steps (see the neigh_modify command). This may be too infrequent for style *dpd* simulations since particles move rapidly and can overlap by large amounts. If this setting yields a non−zero number of "dangerous" reneighborings (printed at the end of a simulation), you should experiment with forcing reneighboring more often and see if system energies/trajectories change.

These pair styles requires you to use the communicate vel yes option so that velocites are stored by ghost atoms.

Related commands:

pair_coeff, fix nvt, fix langevin

Default: none

(Groot) Groot and Warren, J Chem Phys, 107, 4423−35 (1997).

pair_style dsmc command

Syntax:

```
pair_style dsmc max_cell_size seed weighting Tref Nrecompute Nsample
```

- max_cell_size = global maximum cell size for DSMC interactions (distance units)
- seed = random # seed (positive integer)
- weighting = macroparticle weighting
- Tref = reference temperature (temperature units)
- Nrecompute = recompute v*sigma_max every this many timesteps (timesteps)
- Nsample = sample this many times in recomputing v*sigma_max

Examples:

```
pair_style dsmc 2.5 34387 10 1.0 100 20
pair_coeff * * 1.0
pair_coeff 1 1 1.0
```

Description:

Style *dsmc* computes collisions between pairs of particles for a direct simulation Monte Carlo (DSMC) model following the exposition in (Bird). Each collision resets the velocities of the two particles involved. The number of pairwise collisions for each pair or particle types and the length scale within which they occur are determined by the parameters of the pair_style and pair_coeff commands.

Stochastic collisions are performed using the variable hard sphere (VHS) approach, with the user−defined *max_cell_size* value used as the maximum DSMC cell size, and reference cross−sections for collisions given using the pair_coeff command.

There is no pairwise energy or virial contributions associated with this pair style.

The following coefficient must be defined for each pair of atoms types via the pair_coeff command as in the examples above, or in the data file or restart files read by the read_data or read_restart commands:

- sigma (area units, i.e. distance−squared)

The global DSMC *max_cell_size* determines the maximum cell length used in the DSMC calculation. A structured mesh is overlayed on the simulation box such that an integer number of cells are created in each direction for each processor's sub−domain. Cell lengths are adjusted up to the user−specified maximum cell size.

To perform a DSMC simulation with LAMMPS, several additional options should be set in your input script, though LAMMPS does not check for these settings.

Since this pair style does not compute particle forces, you should use the "fix nve/noforce" time integration fix for the DSMC particles, e.g.

```
fix 1 all nve/noforce
```

This pair style assumes that all particles will communicated to neighboring processors every timestep as they move. This makes it possible to perform all collisions between pairs of particles that are on the same processor.

To ensure this occurs, you should use these commands:

```
neighbor 0.0 bin
neigh_modify every 1 delay 0 check no
communicate single cutoff 0.0
```

These commands insure that LAMMPS communicates particles to neighboring processors every timestep and that no ghost atoms are created. The output statistics for a simulation run should indicate there are no ghost particles or neighbors.

Mixing, shift, table, tail correction, restart, rRESPA info:

This pair style does not support mixing. Thus, coefficients for all I,J pairs must be specified explicitly.

This pair style does not support the pair_modify shift option for the energy of the pair interaction.

The pair_modify table option is not relevant for this pair style.

This pair style does not support the pair_modify tail option for adding long−range tail corrections to energy and pressure.

This pair style writes its information to binary restart files, so pair_style and pair_coeff commands do not need to be specified in an input script that reads a restart file. Note that the user−specified random number seed is stored in the restart file, so when a simulation is restarted, each processor will re−initialize its random number generator the same way it did initially. This means the random forces will be random, but will not be the same as they would have been if the original simulation had continued past the restart time.

This pair style can only be used via the *pair* keyword of the run_style respa command. It does not support the *inner*, *middle*, *outer* keywords.

Restrictions:

This style is part of the "mc" package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

Related commands:

pair_coeff, fix nve/noforce, neigh_modify, neighbor, communicate

Default: none

(Bird) G. A. Bird, "Molecular Gas Dynamics and the Direct Simulation of Gas Flows" (1994).

pair_style eam command

pair_style eam/cuda command

pair_style eam/opt command

pair_style eam/alloy command

pair_style eam/alloy/cuda command

pair_style eam/alloy/opt command

pair_style eam/cd command

pair_style eam/fs command

pair_style eam/fs/cuda command

pair_style eam/fs/opt command

Syntax:

```
pair_style style
```

- style = *eam* or *eam/alloy* or *eam/cd* or *eam/fs*

Examples:

```
pair_style eam
pair_coeff * * cuu3
pair_coeff 1*3 1*3 niu3.eam

pair_style eam/alloy
pair_coeff * * ../potentials/NiAlH_jea.eam.alloy Ni Al Ni Ni

pair_style eam/cd
pair_coeff * * ../potentials/FeCr.cdeam Fe Cr

pair_style eam/fs
pair_coeff * * NiAlH_jea.eam.fs Ni Al Ni Ni
```

Description:

Style *eam* computes pairwise interactions for metals and metal alloys using embedded−atom method (EAM) potentials (Daw). The total energy Ei of an atom I is given by

$$E_i = F_\alpha \left(\sum_{j \neq i} \rho_\alpha(r_{ij}) \right) + \frac{1}{2} \sum_{j \neq i} \phi_{\alpha\beta}(r_{ij})$$

where F is the embedding energy which is a function of the atomic electron density rho, phi is a pair potential interaction, and alpha and beta are the element types of atoms I and J. The multi−body nature of the EAM potential is a result of the embedding energy term. Both summations in the formula are over all neighbors J of atom I within the cutoff distance.

The cutoff distance and the tabulated values of the functionals F, rho, and phi are listed in one or more files which are specified by the pair_coeff command. These are ASCII text files in a DYNAMO−style format which is described below. DYNAMO was the original serial EAM MD code, written by the EAM originators. Several DYNAMO potential files for different metals are included in the "potentials" directory of the LAMMPS distribution. All of these files are parameterized in terms of LAMMPS metal units.

IMPORTANT NOTE: The *eam* style reads single−element EAM potentials in the DYNAMO *funcfl* format. Either single element or alloy systems can be modeled using multiple *funcfl* files and style *eam*. For the alloy case LAMMPS mixes the single−element potentials to produce alloy potentials, the same way that DYNAMO does. Alternatively, a single DYNAMO *setfl* file or Finnis/Sinclair EAM file can be used by LAMMPS to model alloy systems by invoking the *eam/alloy* or *eam/cd* or *eam/fs* styles as described below. These files require no mixing since they specify alloy interactions explicitly.

Note that unlike for other potentials, cutoffs for EAM potentials are not set in the pair_style or pair_coeff command; they are specified in the EAM potential files themselves. Likewise, the EAM potential files list atomic masses; thus you do not need to use the mass command to specify them.

There are several WWW sites that distribute and document EAM potentials stored in DYNAMO or other formats:

```
http://www.ctcms.nist.gov/potentials
http://cst−www.nrl.navy.mil/ccm6/ap
http://enpub.fulton.asu.edu/cms/potentials/main/main.htm
```

These potentials should be usable with LAMMPS, though the alternate formats would need to be converted to the DYNAMO format used by LAMMPS and described on this page. The NIST site is maintained by Chandler Becker (cbecker at nist.gov) who is good resource for info on interatomic potentials and file formats.

For style *eam*, potential values are read from a file that is in the DYNAMO single−element *funcfl* format. If the DYNAMO file was created by a Fortran program, it cannot have "D" values in it for exponents. C only recognizes "e" or "E" for scientific notation.

Note that unlike for other potentials, cutoffs for EAM potentials are not set in the pair_style or pair_coeff command; they are specified in the EAM potential files themselves.

For style *eam* a potential file must be assigned to each I,I pair of atom types by using one or more pair_coeff commands, each with a single argument:

- filename

Thus the following command

```
pair_coeff *2 1*2 cuu3.eam
```

will read the cuu3 potential file and use the tabulated Cu values for F, phi, rho that it contains for type pairs 1,1 and 2,2 (type pairs 1,2 and 2,1 are ignored). In effect, this makes atom types 1 and 2 in LAMMPS be Cu atoms. Different single−element files can be assigned to different atom types to model an alloy system. The mixing to create alloy potentials for type pairs with I != J is done automatically the same way that the serial DYNAMO code originally did it; you do not need to specify coefficients for these type pairs.

Funcfl files in the *potentials* directory of the LAMMPS distribution have an ".eam" suffix. A DYNAMO single−element *funcfl* file is formatted as follows:

- line 1: comment (ignored)
- line 2: atomic number, mass, lattice constant, lattice type (e.g. FCC)
- line 3: Nrho, drho, Nr, dr, cutoff

On line 2, all values but the mass are ignored by LAMMPS. The mass is in mass units, e.g. mass number or grams/mole for metal units. The cubic lattice constant is in Angstroms. On line 3, Nrho and Nr are the number of tabulated values in the subsequent arrays, drho and dr are the spacing in density and distance space for the values in those arrays, and the specified cutoff becomes the pairwise cutoff used by LAMMPS for the potential. The units of dr are Angstroms; I'm not sure of the units for drho − some measure of electron density.

Following the three header lines are three arrays of tabulated values:

- embedding function F(rho) (Nrho values)
- effective charge function Z(r) (Nr values)
- density function rho(r) (Nr values)

The values for each array can be listed as multiple values per line, so long as each array starts on a new line. For example, the individual Z(r) values are for r = 0,dr,2*dr, ... (Nr−1)*dr.

The units for the embedding function F are eV. The units for the density function rho are the same as for drho (see above, electron density). The units for the effective charge Z are "atomic charge" or sqrt(Hartree * Bohr−radii). For two interacting atoms i,j this is used by LAMMPS to compute the pair potential term in the EAM energy expression as r*phi, in units of eV−Angstroms, via the formula

```
r*phi = 27.2 * 0.529 * Zi * Zj
```

where 1 Hartree = 27.2 eV and 1 Bohr = 0.529 Angstroms.

Style *eam/alloy* computes pairwise interactions using the same formula as style *eam*. However the associated pair_coeff command reads a DYNAMO *setfl* file instead of a *funcfl* file. *Setfl* files can be used to model a single−element or alloy system. In the alloy case, as explained above, *setfl* files contain explicit tabulated values for alloy interactions. Thus they allow more generality than *funcfl* files for modeling alloys.

For style *eam/alloy*, potential values are read from a file that is in the DYNAMO multi−element *setfl* format, except that element names (Ni, Cu, etc) are added to one of the lines in the file. If the DYNAMO file was created by a Fortran program, it cannot have "D" values in it for exponents. C only recognizes "e" or "E" for scientific notation.

Only a single pair_coeff command is used with the *eam/alloy* style which specifies a DYNAMO *setfl* file, which contains information for M elements. These are mapped to LAMMPS atom types by specifying N additional arguments after the filename in the pair_coeff command, where N is the number of LAMMPS atom types:

- filename

- N element names = mapping of *setfl* elements to atom types

As an example, the potentials/NiAlH_jea.eam.alloy file is a *setfl* file which has tabulated EAM values for 3 elements and their alloy interactions: Ni, Al, and H. If your LAMMPS simulation has 4 atoms types and you want the 1st 3 to be Ni, and the 4th to be Al, you would use the following pair_coeff command:

```
pair_coeff * * NiAlH_jea.eam.alloy Ni Ni Ni Al
```

The 1st 2 arguments must be * * so as to span all LAMMPS atom types. The first three Ni arguments map LAMMPS atom types 1,2,3 to the Ni element in the *setfl* file. The final Al argument maps LAMMPS atom type 4 to the Al element in the *setfl* file. Note that there is no requirement that your simulation use all the elements specified by the *setfl* file.

If a mapping value is specified as NULL, the mapping is not performed. This can be used when an *eam/alloy* potential is used as part of the *hybrid* pair style. The NULL values are placeholders for atom types that will be used with other potentials.

Setfl files in the *potentials* directory of the LAMMPS distribution have an ".eam.alloy" suffix. A DYNAMO multi−element *setfl* file is formatted as follows:

- lines 1,2,3 = comments (ignored)
- line 4: Nelements Element1 Element2 ... ElementN
- line 5: Nrho, drho, Nr, dr, cutoff

In a DYNAMO *setfl* file, line 4 only lists Nelements = the # of elements in the *setfl* file. For LAMMPS, the element name (Ni, Cu, etc) of each element must be added to the line, in the order the elements appear in the file.

The meaning and units of the values in line 5 is the same as for the *funcfl* file described above. Note that the cutoff (in Angstroms) is a global value, valid for all pairwise interactions for all element pairings.

Following the 5 header lines are Nelements sections, one for each element, each with the following format:

- line 1 = atomic number, mass, lattice constant, lattice type (e.g. FCC)
- embedding function F(rho) (Nrho values)
- density function rho(r) (Nr values)

As with the *funcfl* files, only the mass (in mass units, e.g. mass number or grams/mole for metal units) is used by LAMMPS from the 1st line. The cubic lattice constant is in Angstroms. The F and rho arrays are unique to a single element and have the same format and units as in a *funcfl* file.

Following the Nelements sections, Nr values for each pair potential phi(r) array are listed for all i,j element pairs in the same format as other arrays. Since these interactions are symmetric (i,j = j,i) only phi arrays with i >= j are listed, in the following order: i,j = (1,1), (2,1), (2,2), (3,1), (3,2), (3,3), (4,1), ..., (Nelements, Nelements). Unlike the effective charge array Z(r) in *funcfl* files, the tabulated values for each phi function are listed in *setfl* files directly as r*phi (in units of eV−Angstroms), since they are for atom pairs.

Style *eam/cd* is similar to the *eam/alloy* style, except that it computes alloy pairwise interactions using the concentration−dependent embedded−atom method (CD−EAM). This model can reproduce the enthalpy of mixing of alloys over the full composition range, as described in (Stukowski).

The pair_coeff command is specified the same as for the *eam/alloy* style. However the DYNAMO *setfl* file must has two lines added to it, at the end of the file:

- line 1: Comment line (ignored)
- line 2: N Coefficient0 Coefficient1 ... CoeffincientN

The last line begins with the degree *N* of the polynomial function *h(x)* that modifies the cross interaction between A and B elements. Then *N+1* coefficients for the terms of the polynomial are then listed.

Modified EAM *setfl* files used with the *eam/cd* style must contain exactly two elements, i.e. in the current implementation the *eam/cd* style only supports binary alloys. The first and second elements in the input EAM file are always taken as the *A* and *B* species.

CD−EAM files in the *potentials* directory of the LAMMPS distribution have a ".cdeam" suffix.

Style *eam/fs* computes pairwise interactions for metals and metal alloys using a generalized form of EAM potentials due to Finnis and Sinclair (Finnis). The total energy Ei of an atom I is given by

$$E_i = F_\alpha \left(\sum_{j \neq i} \rho_{\alpha\beta}(r_{ij}) \right) + \frac{1}{2} \sum_{j \neq i} \phi_{\alpha\beta}(r_{ij})$$

This has the same form as the EAM formula above, except that rho is now a functional specific to the atomic types of both atoms I and J, so that different elements can contribute differently to the total electron density at an atomic site depending on the identity of the element at that atomic site.

The associated pair_coeff command for style *eam/fs* reads a DYNAMO *setfl* file that has been extended to include additional rho_alpha_beta arrays of tabulated values. A discussion of how FS EAM differs from conventional EAM alloy potentials is given in (Ackland1). An example of such a potential is the same author's Fe−P FS potential (Ackland2). Note that while FS potentials always specify the embedding energy with a square root dependence on the total density, the implementation in LAMMPS does not require that; the user can tabulate any functional form desired in the FS potential files.

For style *eam/fs*, the form of the pair_coeff command is exactly the same as for style *eam/alloy*, e.g.

```
pair_coeff * * NiAlH_jea.eam.fs Ni Ni Ni Al
```

where there are N additional arguments after the filename, where N is the number of LAMMPS atom types. The N values determine the mapping of LAMMPS atom types to EAM elements in the file, as described above for style *eam/alloy*. As with *eam/alloy*, if a mapping value is NULL, the mapping is not performed. This can be used when an *eam/fs* potential is used as part of the *hybrid* pair style. The NULL values are used as placeholders for atom types that will be used with other potentials.

FS EAM files include more information than the DYNAMO *setfl* format files read by *eam/alloy*, in that i,j density functionals for all pairs of elements are included as needed by the Finnis/Sinclair formulation of the EAM.

FS EAM files in the *potentials* directory of the LAMMPS distribution have an ".eam.fs" suffix. They are formatted as follows:

- lines 1,2,3 = comments (ignored)
- line 4: Nelements Element1 Element2 ... ElementN
- line 5: Nrho, drho, Nr, dr, cutoff

The 5−line header section is identical to an EAM *setfl* file.

Following the header are Nelements sections, one for each element I, each with the following format:

- line 1 = atomic number, mass, lattice constant, lattice type (e.g. FCC)
- embedding function F(rho) (Nrho values)
- density function rho(r) for element I at element 1 (Nr values)
- density function rho(r) for element I at element 2
- ...
- density function rho(r) for element I at element Nelement

The units of these quantities in line 1 are the same as for *setfl* files. Note that the rho(r) arrays in Finnis/Sinclair can be asymmetric (i,j != j,i) so there are Nelements^2 of them listed in the file.

Following the Nelements sections, Nr values for each pair potential phi(r) array are listed in the same manner (r*phi, units of eV−Angstroms) as in EAM *setfl* files. Note that in Finnis/Sinclair, the phi(r) arrays are still symmetric, so only phi arrays for i >= j are listed.

Styles with a *cuda*, *gpu*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in this section of the manual. The accelerated styles take the same arguments and should produce the same results, except for round−off and precision issues.

These accelerated styles are part of the "user−cuda", "gpu", and "opt" packages respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the −suffix command−line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See this section of the manual for more instructions on how to use the accelerated styles effectively.

Mixing, shift, table, tail correction, restart, rRESPA info:

For atom type pairs I,J and I != J, where types I and J correspond to two different element types, mixing is performed by LAMMPS as described above with the individual styles. You never need to specify a pair_coeff command with I != J arguments for the eam styles.

This pair style does not support the pair_modify shift, table, and tail options.

The eam pair styles do not write their information to binary restart files, since it is stored in tabulated potential files. Thus, you need to re−specify the pair_style and pair_coeff commands in an input script that reads a restart file.

The eam pair styles can only be used via the *pair* keyword of the run_style respa command. They do not support the *inner*, *middle*, *outer* keywords.

Restrictions:

All of these styles except the *eam/cd* style are part of the "manybody" package. They are only enabled if LAMMPS was built with that package (which it is by default). See the Making LAMMPS section for more info.

The *eam/cd* style is part of the "user−misc" package and also requires the "manybody" package. It is only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

Related commands:

pair_coeff

Default: none

(Ackland1) Ackland, Condensed Matter (2005).

(Ackland2) Ackland, Mendelev, Srolovitz, Han and Barashev, Journal of Physics: Condensed Matter, 16, S2629 (2004).

(Daw) Daw, Baskes, Phys Rev Lett, 50, 1285 (1983). Daw, Baskes, Phys Rev B, 29, 6443 (1984).

(Finnis) Finnis, Sinclair, Philosophical Magazine A, 50, 45 (1984).

(Stukowski) Stukowski, Sadigh, Erhart, Caro; Modeling Simulation Materials Science &Engineering, 7, 075005 (2009).

pair_style eff/cut command

Syntax:

pair_style eff/cut cutoff eradius_limit_flag pressure_flag

- cutoff = global cutoff for Coulombic interactions
- eradius_limit_flag = 0 or 1 for whether electron size is restrained (optional)
- pressure_flag = 0 or 1 to define the type of pressure calculation (optional)

Examples:

```
pair_style eff/cut 39.7
pair_style eff/cut 40.0 1 1
pair_coeff * *
pair_coeff 2 2 20.0
```

Description:

This pair style contains a LAMMPS implementation of the electron Force Field (eFF) potential currently under development at Caltech, as described in (Jaramillo−Botero). The eFF was first introduced by (Su) in 2007.

eFF can be viewed as an approximation to QM wave packet dynamics and Fermionic molecular dynamics, combining the ability of electronic structure methods to describe atomic structure, bonding, and chemistry in materials, and of plasma methods to describe nonequilibrium dynamics of large systems with a large number of highly excited electrons. Yet, eFF relies on a simplification of the electronic wavefunction in which electrons are described as floating Gaussian wave packets whose position and size respond to the various dynamic forces between interacting classical nuclear particles and spherical Gaussian electron wavepackets. The wavefunction is taken to be a Hartree product of the wave packets. To compensate for the lack of explicit antisymmetry in the resulting wavefunction, a spin−dependent Pauli potential is included in the Hamiltonian. Substituting this wavefunction into the time−dependent Schrodinger equation produces equations of motion that correspond − to second order − to classical Hamiltonian relations between electron position and size, and their conjugate momenta. The N−electron wavefunction is described as a product of one−electron Gaussian functions, whose size is a dynamical variable and whose position is not constrained to a nuclear center. This form allows for straightforward propagation of the wavefunction, with time, using a simple formulation from which the equations of motion are then integrated with conventional MD algorithms. In addition to this spin−dependent Pauli repulsion potential term between Gaussians, eFF includes the electron kinetic energy from the Gaussians. These two terms are based on first−principles quantum mechanics. On the other hand, nuclei are described as point charges, which interact with other nuclei and electrons through standard electrostatic potential forms.

The full Hamiltonian (shown below), contains then a standard description for electrostatic interactions between a set of delocalized point and Gaussian charges which include, nuclei−nuclei (NN), electron−electron (ee), and nuclei−electron (Ne). Thus, eFF is a mixed QM−classical mechanics method rather than a conventional force field method (in which electron motions are averaged out into ground state nuclear motions, i.e a single electronic state, and particle interactions are described via empirically parameterized interatomic potential functions). This makes eFF uniquely suited to simulate materials over a wide range of temperatures and pressures where electronically excited and ionized states of matter can occur and coexist. Furthermore, the interactions between particles −nuclei and electrons− reduce to the sum of a set of effective pairwise potentials in the eFF formulation. The *eff/cut* style computes the pairwise Coulomb interactions between nuclei and electrons (E_NN,E_Ne,E_ee), and the quantum−derived Pauli (E_PR) and Kinetic energy interactions potentials between electrons (E_KE) for a total energy expression given as,

$$U\left(R,r,s\right)=E_{NN}\left(R\right)+E_{Ne}\left(R,r,s\right)+E_{ee}\left(r,s\right)+E_{KE}\left(r,s\right)+E_{PR}\left(\uparrow\downarrow,S\right)$$

The individual terms are defined as follows:

$$E_{KE}=\frac{\hbar^2}{m_e}\sum_i\frac{3}{2s_i^2}$$

$$E_{NN}=\frac{1}{4\pi\varepsilon_0}\sum_{i<j}\frac{Z_iZ_j}{R_{ij}}$$

$$E_{Ne}=-\frac{1}{4\pi\varepsilon_0}\sum_{i,j}\frac{Z_i}{R_{ij}}Erf\left(\frac{\sqrt{2}R_{ij}}{s_j}\right)$$

$$E_{ee}=\frac{1}{4\pi\varepsilon_0}\sum_{i<j}\frac{1}{r_{ij}}Erf\left(\frac{\sqrt{2}r_{ij}}{\sqrt{s_i^2+s_j^2}}\right)$$

$$E_{Pauli}=\sum_{\sigma_i=\sigma_j}E\left(\uparrow\uparrow\right)_{ij}+\sum_{\sigma_i\neq\sigma_j}E\left(\uparrow\downarrow\right)_{ij}$$

where, s_i correspond to the electron sizes, the sigmas i's to the fixed spins of the electrons, Z_i to the charges on the nuclei, R_ij to the distances between the nuclei or the nuclei and electrons, and r_ij to the distances between electrons. For additional details see (Jaramillo−Botero).

The overall electrostatics energy is given in Hartree units of energy by default and can be modified by an energy−conversion constant, according to the units chosen (see electron_units). The cutoff Rc, given in Bohrs (by default), truncates the interaction distance. The recommended cutoff for this pair style should follow the minimum image criterion, i.e. half of the minimum unit cell length.

Style *eff/long* (not yet available) computes the same interactions as style *eff/cut* except that an additional damping factor is applied so it can be used in conjunction with the kspace_style command and its *ewald* or *pppm* option. The Coulombic cutoff specified for this style means that pairwise interactions within this distance are computed directly; interactions outside that distance are computed in reciprocal space.

This potential is designed to be used with atom_style electron definitions, in order to handle the description of systems with interacting nuclei and explicit electrons.

The following coefficients must be defined for each pair of atoms types via the pair_coeff command as in the examples above, or in the data file or restart files read by the read_data or read_restart commands, or by mixing as described below:

- cutoff (distance units)

For *eff/cut*, the cutoff coefficient is optional. If it is not used (as in some of the examples above), the default global value specified in the pair_style command is used.

For *eff/long* (not yet available) no cutoff will be specified for an individual I,J type pair via the pair_coeff command. All type pairs use the same global cutoff specified in the pair_style command.

The *eradius_limit_flag* and *pressure_flag* settings are optional. Neither or both must be specified. If not specified they are both set to 0 by default.

The *eradius_limit_flag* is used to restrain electrons from becoming unbounded in size at very high temperatures were the Gaussian wave packet representation breaks down, and from expanding as free particles to infinite size. A setting of 0 means do not impose this restraint. A setting of 1 imposes the restraint. The restraining harmonic potential takes the form E = 1/2k_ss^2 for s > L_box/2, where k_s = 1 Hartrees/Bohr^2.

The *pressure_flag* is used to control between two types of pressure computation: if set to 0, the computed pressure does not include the electronic radial virials contributions to the total pressure (scalar or tensor). If set to 1, the computed pressure will include the electronic radial virial contributions to the total pressure (scalar and tensor).

IMPORTANT NOTE: there are two different pressures that can be reported for eFF when defining this pair_style, one (default) that considers electrons do not contribute radial virial components (i.e. electrons treated as incompressible 'rigid' spheres) and one that does. The radial electronic contributions to the virials are only tallied if the flexible pressure option is set, and this will affect both global and per−atom quantities. In principle, the true pressure of a system is somewhere in between the rigid and the flexible eFF pressures, but, for most cases, the difference between these two pressures will not be significant over long−term averaged runs (i.e. even though the energy partitioning changes, the total energy remains similar).

IMPORTANT NOTE: The currently implemented eFF gives a reasonably accurate description for systems containing nuclei from Z = 1−6. Users interested in applying eFF should restrict to systems where electrons are s−like, or contain p character only insofar as a single lobe of electron density is shifted away from the nuclear center. See further details about some of the virtues and current limitations of the method in (Jaramillo−Botero).

Work is underway to extend the eFF to higher Z elements with increasingly non−spherical electrons (p−block and d−block), to provide explicit terms for electron correlation/exchange, and to improve its computational efficiency via atom models with fixed 2 s core electrons and atom models represented as pseudo−cores plus valence electrons.

The current version adds support for models with fixed−core and effective pseudo−core (i.e. effective core pseudopotentials, ECP) definitions. to enable larger timesteps (i.e. by avoiding the high frequency vibrational modes −translational and radial− of the 2 s electrons), and in the ECP case to reduce the p−character effects in higher Z elements (e.g. Silicon). A fixed−core should be defined with a mass that includes the corresponding nuclear mass plus the 2 s electrons in atomic mass units (2x5.4857990943e−4), and a radius equivalent to that of minimized 1s electrons (see examples under /examples/USER/eff/fixed−core). An pseudo−core should be described with a mass that includes the corresponding nuclear mass, plus all the core electrons (i.e no outer shell electrons), and a radius equivalent to that of a corresponding minimized full−electron system. The charge for a pseudo−core atom should be given by the number of outer shell electrons.

In general, eFF excels at computing the properties of materials in extreme conditions and tracing the system dynamics over multi−picosend timescales; this is particularly relevant where electron excitations can change significantly the nature of bonding in the system. It can capture with surprising accuracy the behavior of such systems because it describes consistently and in an unbiased manner many different kinds of bonds, including covalent, ionic, multicenter, ionic, and plasma, and how they interconvert and/or change when they become excited. eFF also excels in computing the relative thermochemistry of isodemic reactions and conformational changes, where the bonds of the reactants are of the same type as the bonds of the products. eFF assumes that kinetic energy differences dominate the overall exchange energy, which is true when the electrons present are nearly spherical and nodeless and valid for covalent compounds such as dense hydrogen, hydrocarbons, and

diamond; alkali metals (e.g. lithium), alkali earth metals (e.g. beryllium) and semimetals such as boron; and various compounds containing ionic and/or multicenter bonds, such as boron dihydride.

Mixing, shift, table, tail correction, restart, rRESPA info:

For atom type pairs I,J and I != J, the cutoff distance for the *eff/cut* style can be mixed. The default mix value is *geometric*. See the "pair_modify" command for details.

The pair_modify shift option is not relevant for these pair styles.

The *eff/long* (not yet available) style supports the pair_modify table option for tabulation of the short−range portion of the long−range Coulombic interaction.

These pair styles do not support the pair_modify tail option for adding long−range tail corrections to energy and pressure.

These pair styles write their information to binary restart files, so pair_style and pair_coeff commands do not need to be specified in an input script that reads a restart file.

These pair styles can only be used via the *pair* keyword of the run_style respa command. They do not support the *inner*, *middle*, *outer* keywords.

Restrictions:

These pair styles will only be enabled if LAMMPS is built with the "user−eff" package. It will only be enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

These pair styles require that particles store electron attributes such as radius, radial velocity, and radital force, as defined by the atom_style. The *electron* atom style does all of this.

Thes pair styles require you to use the communicate vel yes option so that velocites are stored by ghost atoms.

Related commands:

pair_coeff

Default:

If not specified, eradius_limit_flag = 0 and pressure_flag = 0.

(Su) Su and Goddard, Excited Electron Dynamics Modeling of Warm Dense Matter, Phys Rev Lett, 99:185003 (2007).

(Jaramillo−Botero) Jaramillo−Botero, Su, Qi, Goddard, Large−scale, Long−term Non−adiabatic Electron Molecular Dynamics for Describing Material Properties and Phenomena in Extreme Environments, J Comp Chem, 32, 497−512 (2011).

pair_style eim command

Syntax:

```
pair_style style
```

- style = *eim*

Examples:

```
pair_style eim
pair_coeff * * Na Cl ../potentials/ffield.eim Na Cl
pair_coeff * * Na Cl ffield.eim  Na Na Na Cl
pair_coeff * * Na Cl ../potentials/ffield.eim Cl NULL Na
```

Description:

Style *eim* computes pairwise interactions for ionic compounds using embedded−ion method (EIM) potentials (Zhou). The energy of the system E is given by

$$E = \frac{1}{2}\sum_{i=1}^{N}\sum_{j=i_1}^{i_N}\phi_{ij}\left(r_{ij}\right) + \sum_{i=1}^{N}E_i\left(q_i,\sigma_i\right)$$

The first term is a double pairwise sum over the J neighbors of all I atoms, where phi_ij is a pair potential. The second term sums over the embedding energy E_i of atom I, which is a function of its charge q_i and the electrical potential sigma_i at its location. E_i, q_i, and sigma_i are calculated as

$$\begin{aligned} q_i &= \sum_{j=i_1}^{i_N}\eta_{ji}\left(r_{ij}\right) \\ \sigma_i &= \sum_{j=i_1}^{i_N} q_j\cdot\psi_{ij}\left(r_{ij}\right) \\ E_i\left(q_i,\sigma_i\right) &= \frac{1}{2}\cdot q_i\cdot\sigma_i \end{aligned}$$

where eta_ji is a pairwise function describing electron flow from atom I to atom J, and psi_ij is another pairwise function. The multi−body nature of the EIM potential is a result of the embedding energy term. A complete list of all the pair functions used in EIM is summarized below

$$\phi_{ij}(r) = \begin{cases} \left[\frac{E_{b,ij}\beta_{ij}}{\beta_{ij}-\alpha_{ij}}\exp\left(-\alpha_{ij}\frac{r-r_{e,ij}}{r_{e,ij}}\right) - \frac{E_{b,ij}\alpha_{ij}}{\beta_{ij}-\alpha_{ij}}\exp\left(-\beta_{ij}\frac{r-r_{e,ij}}{r_{e,ij}}\right)\right] f_c\left(r, r_{e,ij}, r_{c,\phi,ij}\right), & p_{ij} = 1 \\ \left[\frac{E_{b,ij}\beta_{ij}}{\beta_{ij}-\alpha_{ij}}\left(\frac{r_{e,ij}}{r}\right)^{\alpha_{ij}} - \frac{E_{b,ij}\alpha_{ij}}{\beta_{ij}-\alpha_{ij}}\left(\frac{r_{e,ij}}{r}\right)^{\beta_{ij}}\right] f_c\left(r, r_{e,ij}, r_{c,\phi,ij}\right), & p_{ij} = 2 \end{cases}$$

$$\eta_{ji} = A_{\eta,ij}\left(\chi_j - \chi_i\right) f_c\left(r, r_{s,\eta,ij}, r_{c,\eta,ij}\right)$$

$$\psi_{ij}(r) = A_{\psi,ij}\exp\left(-\zeta_{ij} r\right) f_c\left(r, r_{s,\psi,ij}, r_{c,\psi,ij}\right)$$

$$f_c(r, r_p, r_c) = 0.510204 erfc\left[\frac{1.64498\left(2r - r_p - r_c\right)}{r_c - r_p}\right] - 0.010204$$

Here E_b, r_e, r_(c,phi), alpha, beta, A_(psi), zeta, r_(s,psi), r_(c,psi), A_(eta), r_(s,eta), r_(c,eta), chi, and pair function type p are parameters, with subscripts ij indicating the two species of atoms in the atomic pair.

IMPORTANT NOTE: Even though the EIM potential is treating atoms as charged ions, you should not use a LAMMPS atom_style that stores a charge on each atom and thus requires you to assign a charge to each atom, e.g. the *charge* or *full* atom styles. This is because the EIM potential infers the charge on an atom from the equation above for q_i; you do not assign charges explicitly.

All the EIM parameters are listed in a potential file which is specified by the pair_coeff command. This is an ASCII text file in a format described below. The "ffield.eim" file included in the "potentials" directory of the LAMMPS distribution currently includes nine elements Li, Na, K, Rb, Cs, F, Cl, Br, and I. A system with any combination of these elements can be modeled. This file is parameterized in terms of LAMMPS metal units.

Note that unlike other potentials, cutoffs for EIM potentials are not set in the pair_style or pair_coeff command; they are specified in the EIM potential file itself. Likewise, the EIM potential file lists atomic masses; thus you do not need to use the mass command to specify them.

Only a single pair_coeff command is used with the *eim* style which specifies an EIM potential file and the element(s) to extract information for. The EIM elements are mapped to LAMMPS atom types by specifying N additional arguments after the filename in the pair_coeff command, where N is the number of LAMMPS atom types:

- Elem1, Elem2, ...
- EIM potential file
- N element names = mapping of EIM elements to atom types

As an example like one of those above, suppose you want to model a system with Na and Cl atoms. If your LAMMPS simulation has 4 atoms types and you want the 1st 3 to be Na, and the 4th to be Cl, you would use the following pair_coeff command:

```
pair_coeff * * Na Cl ffield.eim Na Na Na Cl
```

The 1st 2 arguments must be * * so as to span all LAMMPS atom types. The filename is the EIM potential file. The Na and Cl arguments (before the file name) are the two elements for which info will be extracted from the potentail file. The first three trailing Na arguments map LAMMPS atom types 1,2,3 to the EIM Na element. The final Cl argument maps LAMMPS atom type 4 to the EIM Cl element.

If a mapping value is specified as NULL, the mapping is not performed. This can be used when an *eim* potential is used as part of the *hybrid* pair style. The NULL values are placeholders for atom types that will be used with other potentials.

The ffield.eim file in the *potentials* directory of the LAMMPS distribution is formated as follows:

Lines starting with # are comments and are ignored by LAMMPS. Lines starting with "global:" include three global values. The first value divides the cations from anions, i.e., any elements with electronegativity above this value are viewed as anions, and any elements with electronegativity below this value are viewed as cations. The second and third values are related to the cutoff function − i.e. the 0.510204, 1.64498, and 0.010204 shown in the above equation can be derived from these values.

Lines starting with "element:" are formatted as follows: name of element, atomic number, atomic mass, electronic negativity, atomic radius (LAMMPS ignores it), ionic radius (LAMMPS ignores it), cohesive energy (LAMMPS ignores it), and q0 (must be 0).

Lines starting with "pair:" are entered as: element 1, element 2, r_(c,phi), r_(c,phi) (redundant for historical reasons), E_b, r_e, alpha, beta, r_(c,eta), A_(eta), r_(s,eta), r_(c,psi), A_(psi), zeta, r_(s,psi), and p.

The lines in the file can be in any order; LAMMPS extracts the info it needs.

Restrictions:

This style is part of the "manybody" package. It is only enabled if LAMMPS was built with that package (which it is by default).

Related commands:

pair_coeff

Default: none

(Zhou) Zhou, submitted for publication (2010). Please contact Xiaowang Zhou (Sandia) for details via email at xzhou at sandia.gov.

pair_style gauss command

Syntax:

```
pair_style gauss cutoff
```

- cutoff = global cutoff for Gauss interactions (distance units)

Examples:

```
pair_style gauss 12.0
pair_coeff * * 1.0 0.9
pair_coeff 1 4 1.0 0.9 10.0
```

Description:

Style *gauss* computes a tethering potential of the form

$$E = A \exp(-Br^2) \qquad r < r_c$$

between an atom and its corresponding tether site which will typically be a frozen atom in the simulation. Rc is the cutoff.

The following coefficients must be defined for each pair of atom types via the pair_coeff command as in the examples above, or in the data file or restart files read by the read_data or read_restart commands:

- A (energy units)
- B (1/distance^2 units)
- cutoff (distance units)

The last coefficient is optional. If not specified, the global cutoff is used.

Mixing, shift, table, tail correction, restart, rRESPA info:

This pair style does not support mixing. Thus, coefficients for all I,J pairs must be specified explicitly.

This pair style does not support the pair_modify shift option. There is no effect due to the Gaussian well beyond the cutoff; hence reasonable cutoffs need to be specified.

The pair_modify table and tail options are not relevant for this pair style.

This pair style does not support the pair_modify table option, since a tabulation capability does not exist for this potential.

This pair style writes its information to binary restart files, so pair_style and pair_coeff commands do not need to be specified in an input script that reads a restart file.

This pair style can only be used via the *pair* keyword of the run_style respa command. It does not support the *inner*, *middle*, *outer* keywords.

This pair style tallies an "occupancy" count of how many Gaussian−well sites have an atom within the distance at which the force is a maximum = sqrt(0.5/b). This quantity can be accessed via the compute pair command as a vector of values of length 1.

To print this quantity to the log file (with a descriptive column heading) the following commands could be included in an input script:

```
compute gauss all pair gauss
variable occ equal c_gauss[1]
thermo_style custom step temp epair v_occ
```

Restrictions: none

Related commands:

pair_coeff

Default: none

pair_style gayberne command

pair_style gayberne/gpu command

Syntax:

```
pair_style gayberne gamma upsilon mu cutoff
```

- gamma = shift for potential minimum (typically 1)
- upsilon = exponent for eta orientation−dependent energy function
- mu = exponent for chi orientation−dependent energy function
- cutoff = global cutoff for interactions (distance units)

Examples:

```
pair_style gayberne 1.0 1.0 1.0 10.0
pair_coeff * * 1.0 1.7 1.7 3.4 3.4 1.0 1.0 1.0
```

Description:

The *gayberne* styles compute a Gay−Berne anisotropic LJ interaction (Berardi) between pairs of ellipsoidal particles or an ellipsoidal and spherical particle via the formulas

$$U(\mathbf{A_1}, \mathbf{A_2}, \mathbf{r_{12}}) = \mathbf{U_r}(\mathbf{A_1}, \mathbf{A_2}, \mathbf{r_{12}}, \gamma) \cdot \eta_{12}(\mathbf{A_1}, \mathbf{A_2}, \upsilon) \cdot \chi_{12}(\mathbf{A_1}, \mathbf{A_2}, \mathbf{r_{12}}, \mu)$$

$$U_r = 4\epsilon(\varrho^{12} - \varrho^{6})$$

$$\varrho = \frac{\sigma}{h_{12} + \gamma\sigma}$$

where A1 and A2 are the transformation matrices from the simulation box frame to the body frame and r12 is the center to center vector between the particles. Ur controls the shifted distance dependent interaction based on the distance of closest approach of the two particles (h12) and the user−specified shift parameter gamma. When both particles are spherical, the formula reduces to the usual Lennard−Jones interaction (see details below for when Gay−Berne treats a particle as "spherical").

For large uniform molecules it has been shown that the energy parameters are approximately representable in terms of local contact curvatures (Everaers):

$$\epsilon_a = \sigma \cdot \frac{a}{b \cdot c}; \epsilon_b = \sigma \cdot \frac{b}{a \cdot c}; \epsilon_c = \sigma \cdot \frac{c}{a \cdot b}$$

The variable names utilized as potential parameters are for the most part taken from (Everaers) in order to be consistent with the RE−squared pair potential. Details on the upsilon and mu parameters are given here.

More details of the Gay−Berne formulation are given in the references listed below and in this supplementary document.

Use of this pair style requires the NVE, NVT, or NPT fixes with the *asphere* extension (e.g. fix nve/asphere) in order to integrate particle rotation. Additionally, atom_style ellipsoid should be used since it defines the rotational state and the size and shape of each ellipsoidal particle.

The following coefficients must be defined for each pair of atoms types via the pair_coeff command as in the examples above, or in the data file or restart files read by the read_data or read_restart commands, or by mixing as described below:

- epsilon = well depth (energy units)
- sigma = minimum effective particle radii (distance units)
- epsilon_i_a = relative well depth of type I for side−to−side interactions
- epsilon_i_b = relative well depth of type I for face−to−face interactions
- epsilon_i_c = relative well depth of type I for end−to−end interactions
- epsilon_j_a = relative well depth of type J for side−to−side interactions
- epsilon_j_b = relative well depth of type J for face−to−face interactions
- epsilon_j_c = relative well depth of type J for end−to−end interactions
- cutoff (distance units)

The last coefficient is optional. If not specified, the global cutoff specified in the pair_style command is used.

It is typical with the Gay−Berne potential to define *sigma* as the minimum of the 3 shape diameters of the particles involved in an I,I interaction, though this is not required. Note that this is a different meaning for *sigma* than the pair_style resquared potential uses.

The epsilon_i and epsilon_j coefficients are actually defined for atom types, not for pairs of atom types. Thus, in a series of pair_coeff commands, they only need to be specified once for each atom type.

Specifically, if any of epsilon_i_a, epsilon_i_b, epsilon_i_c are non−zero, the three values are assigned to atom type I. If all the epsilon_i values are zero, they are ignored. If any of epsilon_j_a, epsilon_j_b, epsilon_j_c are non−zero, the three values are assigned to atom type J. If all three epsilon_i values are zero, they are ignored. Thus the typical way to define the epsilon_i and epsilon_j coefficients is to list their values in "pair_coeff I J" commands when I = J, but set them to 0.0 when I != J. If you do list them when I != J, you should insure they are consistent with their values in other pair_coeff commands.

Note that if this potential is being used as a sub−style of pair_style hybrid, and there is no "pair_coeff I I" setting made for Gay−Berne for a particular type I (because I−I interactions are computed by another hybrid pair potential), then you still need to insure the epsilon a,b,c coefficients are assigned to that type in a "pair_coeff I J" command.

IMPORTANT NOTE: If the epsilon a = b = c for an atom type, and if the shape of the particle itself is spherical, meaning its 3 shape parameters are all the same, then the particle is treated as an LJ sphere by the Gay−Berne potential. This is significant because if two LJ spheres interact, then the simple Lennard−Jones formula is used to compute their interaction energy/force using the specified epsilon and sigma as the standard LJ parameters. This is much cheaper to compute than the full Gay−Berne formula. To treat the particle as a LJ sphere with sigma = D, you should normally set epsilon a = b = c = 1.0, set the pair_coeff sigma = D, and also set the 3 shape parameters for the particle to D. The one exception is that if the 3 shape parameters are set to 0.0, which is a valid way in LAMMPS to specify a point particle, then the Gay−Berne potential will treat that as shape parameters of 1.0 (i.e. a LJ particle with sigma = 1), since it requires finite−size particles. In this case you should still set the pair_coeff sigma to 1.0 as well.

Styles with a *cuda*, *gpu*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in this section of the

manual. The accelerated styles take the same arguments and should produce the same results, except for round−off and precision issues.

These accelerated styles are part of the "user−cuda", "gpu", and "opt" packages respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the −suffix command−line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See this section of the manual for more instructions on how to use the accelerated styles effectively.

Mixing, shift, table, tail correction, restart, rRESPA info:

For atom type pairs I,J and I != J, the epsilon and sigma coefficients and cutoff distance for this pair style can be mixed. The default mix value is *geometric*. See the "pair_modify" command for details.

This pair styles supports the pair_modify shift option for the energy of the Lennard−Jones portion of the pair interaction, but only for sphere−sphere interactions. There is no shifting performed for ellipsoidal interactions due to the anisotropic dependence of the interaction.

The pair_modify table option is not relevant for this pair style.

This pair style does not support the pair_modify tail option for adding long−range tail corrections to energy and pressure.

This pair style writes its information to binary restart files, so pair_style and pair_coeff commands do not need to be specified in an input script that reads a restart file.

This pair style can only be used via the *pair* keyword of the run_style respa command. It does not support the *inner*, *middle*, *outer* keywords.

Restrictions:

The *gayberne* style is part of the "asphere" package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

These pair style require that atoms store torque and a quaternion to represent their orientation, as defined by the atom_style. It also require they store a per−type shape. The particles cannot store a per−particle diameter.

This pair style requires that atoms be ellipsoids as defined by the atom_style ellipsoid command.

Particles acted on by the potential can be extended aspherical or spherical particles, or point particles. Spherical particles have all 3 of their shape parameters equal to each other. Point particles have all 3 of their shape parameters equal to 0.0.

The Gay−Berne potential does not become isotropic as r increases (Everaers). The distance−of−closest−approach approximation used by LAMMPS becomes less accurate when high−aspect ratio ellipsoids are used.

Related commands:

pair_coeff, fix nve/asphere, compute temp/asphere, pair_style resquared

Default: none

(Everaers) Everaers and Ejtehadi, Phys Rev E, 67, 041710 (2003).

(Berardi) Berardi, Fava, Zannoni, Chem Phys Lett, 297, 8−14 (1998). Berardi, Muccioli, Zannoni, J Chem Phys, 128, 024905 (2008).

(Perram) Perram and Rasmussen, Phys Rev E, 54, 6565−6572 (1996).

(Allen) Allen and Germano, Mol Phys 104, 3225−3235 (2006).

pair_style gran/hooke command

pair_style gran/cuda command

pair_style gran/hooke/history command

pair_style gran/hertz/history command

Syntax:

```
pair_style style Kn Kt gamma_n gamma_t xmu dampflag
```

- style = *gran/hooke* or *gran/hooke/history* or *gran/hertz/history*
- Kn = elastic constant for normal particle repulsion (force/distance units or pressure units − see discussion below)
- Kt = elastic constant for tangential contact (force/distance units or pressure units − see discussion below)
- gamma_n = damping coefficient for collisions in normal direction (1/time units or 1/time−distance units − see discussion below)
- gamma_t = damping coefficient for collisions in tangential direction (1/time units or 1/time−distance units − see discussion below)
- xmu = static yield criterion (unitless fraction between 0.0 and 1.0)
- dampflag = 0 or 1 if tangential damping force is excluded or included

IMPORTANT NOTE: Versions of LAMMPS before 9Jan09 had different style names for granular force fields. This is to emphasize the fact that the Hertzian equation has changed to model polydispersity more accurately. A side effect of the change is that the Kn, Kt, gamma_n, and gamma_t coefficients in the pair_style command must be specified with different values in order to reproduce calculations made with earlier versions of LAMMPS, even for monodisperse systems. See the NOTE below for details.

Examples:

```
pair_style gran/hooke/history 200000.0 NULL 50.0 NULL 0.5 1
pair_style gran/hooke 200000.0 70000.0 50.0 30.0 0.5 0
```

Description:

The *gran* styles use the following formulas for the frictional force between two granular particles, as described in (Brilliantov), (Silbert), and (Zhang), when the distance r between two particles of radii Ri and Rj is less than their contact distance d = Ri + Rj. There is no force between the particles when r > d.

The two Hookean styles use this formula:

$$F_{hk} = (k_n \delta \mathbf{n}_{ij} - m_{\mathrm{eff}} \gamma_n \mathbf{v}_n) - (k_t \mathbf{\Delta s}_t + m_{\mathrm{eff}} \gamma_t \mathbf{v}_t)$$

The Hertzian style uses this formula:

$$F_{hz} = \sqrt{\delta}\sqrt{\frac{R_i R_j}{R_i + R_j}}F_{hk} = \sqrt{\delta}\sqrt{\frac{R_i R_j}{R_i + R_j}}\left[(k_n \delta \mathbf{n}_{ij} - m_{\text{eff}}\gamma_n \mathbf{v}_n) - (k_t \mathbf{\Delta s}_t + m_{\text{eff}}\gamma_t \mathbf{v}_t)\right]$$

In both equations the first parenthesized term is the normal force between the two particles and the second parenthesized term is the tangential force. The normal force has 2 terms, a contact force and a damping force. The tangential force also has 2 terms: a shear force and a damping force. The shear force is a "history" effect that accounts for the tangential displacement between the particles for the duration of the time they are in contact. This term is included in pair styles *hooke/history* and *hertz/history*, but is not included in pair style *hooke*. The tangential damping force term is included in all three pair styles if *dampflag* is set to 1; it is not included if *dampflag* is set to 0.

The other quantities in the equations are as follows:

- delta = d − r = overlap distance of 2 particles
- Kn = elastic constant for normal contact
- Kt = elastic constant for tangential contact
- gamma_n = viscoelastic damping constant for normal contact
- gamma_t = viscoelastic damping constant for tangential contact
- m_eff = Mi Mj / (Mi + Mj) = effective mass of 2 particles of mass Mi and Mj
- Delta St = tangential displacement vector between 2 spherical particles which is truncated to satisfy a frictional yield criterion
- n_ij = unit vector along the line connecting the centers of the 2 particles
- Vn = normal component of the relative velocity of the 2 particles
- Vt = tangential component of the relative velocity of the 2 particles

The Kn, Kt, gamma_n, and gamma_t coefficients are specified as parameters to the pair_style command. If a NULL is used for Kt, then a default value is used where Kt = 2/7 Kn. If a NULL is used for gamma_t, then a default value is used where gamma_t = 1/2 gamma_n.

The interpretation and units for these 4 coefficients are different in the Hookean versus Hertzian equations.

The Hookean model is one where the normal push−back force for two overlapping particles is a linear function of the overlap distance. Thus the specified Kn is in units of (force/distance). Note that this push−back force is independent of absolute particle size (in the monodisperse case) and of the relative sizes of the two particles (in the polydisperse case). This model also applies to the other terms in the force equation so that the specified gamma_n is in units of (1/time), Kt is in units of (force/distance), and gamma_t is in units of (1/time).

The Hertzian model is one where the normal push−back force for two overlapping particles is proportional to the area of overlap of the two particles, and is thus a non−linear function of overlap distance. Thus Kn has units of force per area and is thus specified in units of (pressure). The effects of absolute particle size (monodispersity) and relative size (polydispersity) are captured in the radii−dependent pre−factors. When these pre−factors are carried through to the other terms in the force equation it means that the specified gamma_n is in units of (1/(time*distance)), Kt is in units of (pressure), and gamma_t is in units of (1/(time*distance)).

Note that in the Hookean case, Kn can be thought of as a linear spring constant with units of force/distance. In the Hertzian case, Kn is like a non−linear spring constant with units of force/area or pressure, and as shown in the (Zhang) paper, Kn = 4G / (3(1−nu)) where nu = the Poisson ratio, G = shear modulus = E / (2(1+nu)), and E = Young's modulus. Similarly, Kt = 8G / (2−nu). Thus in the Hertzian case Kn and Kt can be set to values that corresponds to properties of the material being modeled. This is also true in the Hookean case, except that a spring constant must be chosen that is appropriate for the absolute size of particles in the model. Since relative

particle sizes are not accounted for, the Hookean styles may not be a suitable model for polydisperse systems.

IMPORTANT NOTE: In versions of LAMMPS before 9Jan09, the equation for Hertzian interactions did not include the sqrt(RiRj/Ri+Rj) term and thus was not as accurate for polydisperse systems. For monodisperse systems, sqrt(RiRj/Ri+Rj) is a constant factor that effectively scales all 4 coefficients: Kn, Kt, gamma_n, gamma_t. Thus you can set the values of these 4 coefficients appropriately in the current code to reproduce the results of a previous Hertzian monodisperse calculation. For example, for the common case of a monodisperse system with particles of diameter 1, all 4 of these coefficients should now be set 2x larger than they were previously.

Xmu is also specified in the pair_style command and is the upper limit of the tangential force through the Coulomb criterion Ft = xmu*Fn, where Ft and Fn are the total tangential and normal force components in the formulas above. Thus in the Hookean case, the tangential force between 2 particles grows according to a tangential spring and dash−pot model until Ft/Fn = xmu and is then held at Ft = Fn*xmu until the particles lose contact. In the Hertzian case, a similar analogy holds, though the spring is no longer linear.

For granular styles there are no additional coefficients to set for each pair of atom types via the pair_coeff command. All settings are global and are made via the pair_style command. However you must still use the pair_coeff for all pairs of granular atom types. For example the command

```
pair_coeff * *
```

should be used if all atoms in the simulation interact via a granular potential (i.e. one of the pair styles above is used). If a granular potential is used as a sub−style of pair_style hybrid, then specific atom types can be used in the pair_coeff command to determine which atoms interact via a granular potential.

Styles with a *cuda*, *gpu*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in this section of the manual. The accelerated styles take the same arguments and should produce the same results, except for round−off and precision issues.

These accelerated styles are part of the "user−cuda", "gpu", and "opt" packages respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the −suffix command−line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See this section of the manual for more instructions on how to use the accelerated styles effectively.

Mixing, shift, table, tail correction, restart, rRESPA info:

The pair_modify mix, shift, table, and tail options are not relevant for granular pair styles.

These pair styles write their information to binary restart files, so a pair_style command does not need to be specified in an input script that reads a restart file.

These pair styles can only be used via the *pair* keyword of the run_style respa command. They do not support the *inner*, *middle*, *outer* keywords.

Restrictions: none

All the granular pair styles are part of the "granular" package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

These pair styles require that atoms store torque and angular velocity (omega) as defined by the atom_style. They also require a per−particle radius is stored. The *sphere* atom style does all of this.

This pair style requires you to use the communicate vel yes option so that velocites are stored by ghost atoms.

Related commands:

pair_coeff

Default: none

(Brilliantov) Brilliantov, Spahn, Hertzsch, Poschel, Phys Rev E, 53, p 5382−5392 (1996).

(Silbert) Silbert, Ertas, Grest, Halsey, Levine, Plimpton, Phys Rev E, 64, p 051302 (2001).

(Zhang) Zhang and Makse, Phys Rev E, 72, p 011301 (2005).

pair_style lj/gromacs command

pair_style lj/gromacs/cuda command

pair_style lj/gromacs/coul/gromacs command

pair_style lj/gromacs/coul/gromacs/cuda command

Syntax:

```
pair_style style args
```

- style = *lj/gromacs* or *lj/gromacs/coul/gromacs*
- args = list of arguments for a particular style

```
lj/gromacs args = inner outer
    inner, outer = global switching cutoffs for Lennard Jones
  lj/gromacs/coul/gromacs args = inner outer (inner2) (outer2)
    inner, outer = global switching cutoffs for Lennard Jones (and Coulombic if only 2 args)
    inner2, outer2 = global switching cutoffs for Coulombic (optional)
```

Examples:

```
pair_style lj/gromacs 9.0 12.0
pair_coeff * * 100.0 2.0
pair_coeff 2 2 100.0 2.0 8.0 10.0

pair_style lj/gromacs/coul/gromacs 9.0 12.0
pair_style lj/gromacs/coul/gromacs 8.0 10.0 7.0 9.0
pair_coeff * * 100.0 2.0
```

Description:

The *lj/gromacs* styles compute LJ and Coulombic interactions with an additional switching function S(r) that ramps the energy and force smoothly to zero between an inner and outer cutoff. It is a commonly used potential in the GROMACS MD code and for the coarse−grained models of (Marrink).

$$
\begin{aligned}
E_{LJ} &= 4\epsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right] + S_{LJ}(r) \qquad r < r_c \\
E_C &= \frac{Cq_iq_j}{\epsilon r} + S_C(r) \qquad r < r_c \\
S(r) &= 0 \qquad r < r_1 \\
S(r) &= A(r-r_1)^2 + B(r-r_1)^3 \qquad r_1 < r < r_c
\end{aligned}
$$

R1 is the inner cutoff; Rc is the outer cutoff. The coefficients A and B are computed by LAMMPS to perform the smoothing. The function S(r) is actually applied once to each term of the LJ formula and once to the Coulombic formula, so there are 2 or 3 sets of A,B coefficients depending on which pair_style is used. The boundary conditions applied to the smoothing function are as follows: S(r1) = S'(r1) = 0, S(rc) = −F(rc), S'(rc) = −F'(rc),

where F(r) is the correpsonding term in the LJ or Coulombic function and a single quote represents a derivative with respect to r.

The inner and outer cutoff for the LJ and Coulombic terms can be the same or different depending on whether 2 or 4 arguments are used in the pair_style command. The inner LJ cutoff must be > 0, but the inner Coulombic cutoff can be >= 0.

The following coefficients must be defined for each pair of atoms types via the pair_coeff command as in the examples above, or in the data file or restart files read by the read_data or read_restart commands, or by mixing as described below:

- epsilon (energy units)
- sigma (distance units)
- inner (distance units)
- outer (distance units)

Note that sigma is defined in the LJ formula as the zero−crossing distance for the potential, not as the energy minimum at 2^(1/6) sigma.

The last 2 coefficients are optional inner and outer cutoffs for style *lj/gromacs*. If not specified, the global *inner* and *outer* values are used.

The last 2 coefficients cannot be used with style *lj/gromacs/coul/gromacs* because this force field does not allow varying cutoffs for individual atom pairs; all pairs use the global cutoff(s) specified in the pair_style command.

Styles with a *cuda*, *gpu*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in this section of the manual. The accelerated styles take the same arguments and should produce the same results, except for round−off and precision issues.

These accelerated styles are part of the "user−cuda", "gpu", and "opt" packages respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the −suffix command−line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See this section of the manual for more instructions on how to use the accelerated styles effectively.

Mixing, shift, table, tail correction, restart, rRESPA info:

For atom type pairs I,J and I != J, the epsilon and sigma coefficients and cutoff distance for all of the lj/cut pair styles can be mixed. The default mix value is *geometric*. See the "pair_modify" command for details.

None of the GROMACS pair styles support the pair_modify shift option, since the Lennard−Jones portion of the pair interaction is already smoothed to 0.0 at the cutoff.

The pair_modify table option is not relevant for this pair style.

None of the GROMACS pair styles support the pair_modify tail option for adding long−range tail corrections to energy and pressure, since there are no corrections for a potential that goes to 0.0 at the cutoff.

All of the GROMACS pair styles write their information to binary restart files, so pair_style and pair_coeff commands do not need to be specified in an input script that reads a restart file.

All of the GROMACS pair styles can only be used via the *pair* keyword of the run_style respa command. They do not support the *inner*, *middle*, *outer* keywords.

Restrictions: none

Related commands:

pair_coeff

Default: none

(Marrink) Marrink, de Vries, Mark, J Phys Chem B, 108, 750−760 (2004).

pair_style hbond/dreiding/lj command

pair_style hbond/dreiding/morse command

Syntax:

```
pair_style style N inner_distance_cutoff outer_distance_cutoff angle_cutof
```

- style = *hbond/dreiding/lj* or *hbond/dreiding/morse*
- n = cosine angle periodicity
- inner_distance_cutoff = global inner spline cutoff for Donor−Acceptor interactions (distance units)
- outer_distance_cutoff = global cutoff for Donor−Acceptor interactions (distance units)
- angle_cutoff = global angle cutoff for Acceptor−Hydrogen−Donor
- interactions (degrees)

Examples:

```
pair_style hbond/dreiding/lj 4 4.5 5.0 90
pair_coeff * * 3 i 100.0 3.1
pair_coeff * * 2*5 i 100.0 3.1 2 15.0 20.0 135.0

pair_style hbond/dreiding/morse 2 3.0 4.6 75.0
pair_coeff * * 3 j 100.0 1.0 2.0
pair_coeff * * 2*5 j 100.0 1.0 2.0 4.0 6.0
```

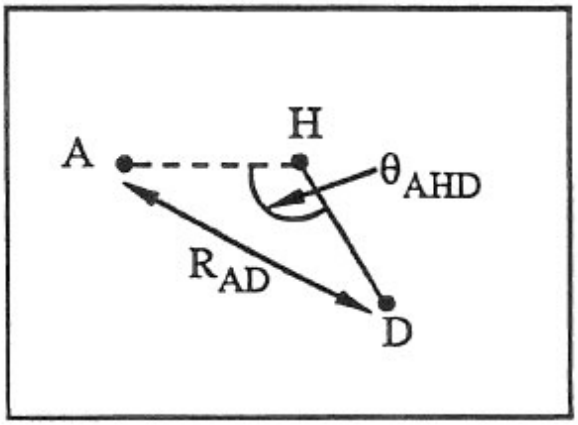
Description:

The *hbond/dreiding* styles compute the Acceptor−Hydrogen−Donor (AHD) 3−body hydrogen bond interaction for the DREIDING force field, given by:

$$
\begin{aligned}
E &= [LJ(r)|Morse(r)] && r < r_{\text{in}} \\
&= S(r) * [LJ(r)|Morse(r)] && r_{\text{in}} < r < r_{\text{out}} \\
&= 0 && r > r_{\text{out}} \\
LJ(r) &= AR^{-12} - BR^{-10} cos^n\theta = \epsilon \left\{ 5 \left[\frac{\sigma}{r}\right]^{12} - 6 \left[\frac{\sigma}{r}\right]^{10} \right\} cos^n\theta \\
Morse(r) &= D_0 \left\{ \chi^2 - 2\chi \right\} cos^n\theta = D_0 \left\{ e^{-2\alpha(r-r_0)} - 2e^{-\alpha(r-r_0)} \right\} cos^n\theta \\
S(r) &= \frac{\left[r_{\text{out}}^2 - r^2\right]^2 \left[r_{\text{out}}^2 + 2r^2 - 3r_{\text{in}}^2\right]}{\left[r_{\text{out}}^2 - {r_{\text{in}}}^2\right]^3}
\end{aligned}
$$

where Rin is the inner spline distance cutoff, Rout is the outer distance cutoff, theta_c is the angle cutoff, and n is the cosine periodicity.

Here, *r* is the radial distance between the donor (D) and acceptor (A) atoms and *theta* is the bond angle between the acceptor, the hydrogen (H) and the donor atoms:



These 3−body interactions can be defined for pairs of acceptor and donor atoms, based on atom types. For each donor/acceptor atom pair, the 3rd atom in the interaction is a hydrogen permanently bonded to the donor atom, e.g. in a bond list read in from a data file via the read_data command. The atom types of possible hydrogen atoms for each donor/acceptor type pair are specified by the pair_coeff command (see below).

Style *hbond/dreiding/lj* is the original DREIDING potential of (Mayo). It uses a LJ 12/10 functional for the Donor−Acceptor interactions. To match the results in the original paper, use n = 4.

Style *hbond/dreiding/morse* is an improved version using a Morse potential for the Donor−Acceptor interactions. (Liu) showed that the Morse form gives improved results for Dendrimer simulations, when n = 2.

See this howto section of the manual for more information on the DREIDING forcefield.

Because the Dreiding hydrogen bond potential is only one portion of an overall force field which typically includes other pairwise interactions, it is common to use it as a sub−style in a pair_style hybrid or hybrid/overlay command.

The following coefficients must be defined for pairs of eligible donor/acceptor types via the pair_coeff command as in the examples above.

IMPORTANT NOTE: Unlike other pair styles and their associated pair_coeff commands, you do not need to specify pair_coeff settings for all possible I,J type pairs. Only I,J type pairs for atoms which act as joint donors/acceptors need to be specified; all other type pairs are assumed to be inactive.

IMPORTANT NOTE: A pair_coeff command can be speficied multiple times for the same donor/acceptor type pair. This enables multiple hydrogen types to be assigned to the same donor/acceptor type pair. For other pair_styles, if the pair_coeff command is re−used for the same I.J type pair, the settings for that type pair are overwritten. For the hydrogen bond potentials this is not the case; the settings are cummulative. This means the only way to turn off a previous setting, is to re−use the pair_style command and start over.

For the *hbond/dreiding/lj* style the list of coefficients is as follows:

- K = hydrogen atom type = 1 to Ntypes
- donor flag = *i* or *j*
- epsilon (energy units)
- sigma (distance units)
- n = exponent in formula above
- distance cutoff (distance units)
- angle cutoff (degrees)

For the *hbond/dreiding/morse* style the list of coefficients is as follows:

- K = hydrogen atom type = 1 to Ntypes
- donor flag = *i* or *j*
- D0 (energy units)
- alpha (1/distance units)
- r0 (distance units)
- n = exponent in formula above
- distance cutoff (distance units)
- angle cutoff (degrees)

A single hydrogen atom type K can be specified, or a wild−card asterisk can be used in place of or in conjunction with the K arguments to select multiple types as hydrogens. This takes the form "*" or "*n" or "n*" or "m*n". See the pair_coeff command doc page for details.

If the donor flag is *i*, then the atom of type I in the pair_coeff command is treated as the donor, and J is the acceptor. If the donor flag is *j*, then the atom of type J in the pair_coeff command is treated as the donor and I is the donor. This option is required because the pair_coeff command requires that I <= J.

Epsilon and sigma are settings for the hydrogen bond potential based on a Lennard−Jones functional form. Note that sigma is defined as the zero−crossing distance for the potential, not as the energy minimum at 2^(1/6) sigma.

D0 and alpha and r0 are settings for the hydrogen bond potential based on a Morse functional form.

The last 3 coefficients for both styles are optional. If not specified, the global n, distance cutoff, and angle cutoff specified in the pair_style command are used. If you wish to only override the 2nd or 3rd optional parameter, you must also specify the preceding optional parameters.

Mixing, shift, table, tail correction, restart, rRESPA info:

These pair styles do not support mixing. You must explicitly identify each donor/acceptor type pair.

These styles do not support the pair_modify shift option for the energy of the interactions.

The pair_modify table option is not relevant for these pair styles.

These pair styles do not support the pair_modify tail option for adding long−range tail corrections to energy and pressure.

These pair styles do not write their information to binary restart files, so pair_style and pair_coeff commands need to be re−specified in an input script that reads a restart file.

These pair styles can only be used via the *pair* keyword of the run_style respa command. They do not support the *inner*, *middle*, *outer* keywords.

These pair styles tally a count of how many hydrogen bonding interactions they calculate each timestep and the hbond energy. These quantities can be accessed via the compute pair command as a vector of values of length 2.

To print these quantities to the log file (with a descriptive column heading) the following commands could be included in an input script:

```
compute hb all pair hbond/dreiding/lj
variable n_hbond equal c_hb[1] #number hbonds
```

```
variable E_hbond equal c_hb[2] #hbond energy
thermo_style custom step temp epair v_E_hbond
```

Restrictions: none

Related commands:

pair_coeff

Default: none

(Mayo) Mayo, Olfason, Goddard III, J Phys Chem, 94, 8897−8909 (1990).

(Liu) Liu, Bryantsev, Diallo, Goddard III, J. Am. Chem. Soc 131 (8) 2798 (2009)

pair_style hybrid command

pair_style hybrid/overlay command

Syntax:

```
pair_style hybrid style1 args style2 args ...
pair_style hybrid/overlay style1 args style2 args ...
```

- style1,style2 = list of one or more pair styles and their arguments

Examples:

```
pair_style hybrid lj/cut/coul/cut 10.0 eam lj/cut 5.0
pair_coeff 1*2 1*2 eam niu3
pair_coeff 3 3 lj/cut/coul/cut 1.0 1.0
pair_coeff 1*2 3 lj/cut 0.5 1.2

pair_style hybrid/overlay lj/cut 2.5 coul/long 2.0
pair_coeff * * lj/cut 1.0 1.0
pair_coeff * * coul/long
```

Description:

The *hybrid* and *hybrid/overlay* styles enable the use of multiple pair styles in one simulation. With the *hybrid* style, exactly one pair style is assigned to each pair of atom types. With the *hybrid/overlay* style, one or more pair styles can be assigned to each pair of atom types. The assignment of pair styles to type pairs is made via the pair_coeff command.

Here are two examples of hybrid simulations. The *hybrid* style could be used for a simulation of a metal droplet on a LJ surface. The metal atoms interact with each other via an *eam* potential, the surface atoms interact with each other via a *lj/cut* potential, and the metal/surface interaction is also computed via a *lj/cut* potential. The *hybrid/overlay* style could be used as in the 2nd example above, where multiple potentials are superposed in an additive fashion to compute the interaction between atoms. In this example, using *lj/cut* and *coul/long* together gives the same result as if the *lj/cut/coul/long* potential were used by itself. In this case, it would be more efficient to use the single combined potential, but in general any combination of pair potentials can be used together in to produce an interaction that is not encoded in any single pair_style file, e.g. adding Coulombic forces between granular particles.

All pair styles that will be used are listed as "sub−styles" following the *hybrid* or *hybrid/overlay* keyword, in any order. Each sub−style's name is followed by its usual arguments, as illustrated in the example above. See the doc pages of individual pair styles for a listing and explanation of the appropriate arguments.

In the pair_coeff commands, the name of a pair style must be added after the I,J type specification, with the remaining coefficients being those appropriate to that style. For example, consider a simulation with 3 atom types: types 1 and 2 are Ni atoms, type 3 are LJ atoms with charges. The following commands would set up a hybrid simulation:

```
pair_style hybrid eam/alloy lj/cut/coul/cut 10.0 lj/cut 8.0
pair_coeff * * eam/alloy nialhjea Ni Ni NULL
pair_coeff 3 3 lj/cut/coul/cut 1.0 1.0
pair_coeff 1*2 3 lj/cut 0.8 1.3
```

If pair coefficients are specified in the data file read via the read_data command, then the same rule applies. E.g. "eam/alloy" or "lj/cut" must be added after the atom type, for each line in the "Pair Coeffs" section, e.g.

```
Pair Coeffs

1 lj/cut/coul/cut 1.0 1.0
...
```

Note that the pair_coeff command for some potentials such as pair_style eam/alloy includes a mapping specification of elements to all atom types, which in the hybrid case, can include atom types not assigned to the *eam/alloy* potential. The NULL keyword is used by many such potentials (eam/alloy, Tersoff, AIREBO, etc), to denote an atom type that will be assigned to a different sub−style.

For the *hybrid* style, each atom type pair I,J is assigned to exactly one sub−style. Just as with a simulation using a single pair style, if you specify the same atom type pair in a second pair_coeff command, the previous assignment will be overwritten.

For the *hybrid/overlay* style, each atom type pair I,J can be assigned to one or more sub−styles. If you specify the same atom type pair in a second pair_coeff command with a new sub−style, then the second sub−style is added to the list of potentials that will be calculated for two interacting atoms of those types. If you specify the same atom type pair in a second pair_coeff command with a sub−style that has already been defined for that pair of atoms, then the new pair coefficients simply override the previous ones, as in the normal usage of the pair_coeff command. E.g. these two sets of commands are the same:

```
pair_style lj/cut 2.5
pair_coeff * * 1.0 1.0
pair_coeff 2 2 1.5 0.8

pair_style hybrid/overlay lj/cut 2.5
pair_coeff * * lj/cut 1.0 1.0
pair_coeff 2 2 lj/cut 1.5 0.8
```

Coefficients must be defined for each pair of atoms types via the pair_coeff command as described above, or in the data file or restart files read by the read_data or read_restart commands, or by mixing as described below.

For both the *hybrid* and *hybrid/overlay* styles, every atom type pair I,J (where I <= J) must be assigned to at least one sub−style via the pair_coeff command as in the examples above, or in the data file read by the read_data, or by mixing as described below.

If you want there to be no interactions between a particular pair of atom types, you have 3 choices. You can assign the type pair to some sub−style and use the neigh_modify exclude type command. You can assign it to some sub−style and set the coefficients so that there is effectively no interaction (e.g. epsilon = 0.0 in a LJ potential). Or, for *hybrid* and *hybrid/overlay* simulations, you can use this form of the pair_coeff command in your input script:

```
pair_coeff      2 3 none
```

or this form in the "Pair Coeffs" section of the data file:

```
3 none
```

If an assignment to *none* is made in a simulation with the *hybrid/overlay* pair style, it wipes out all previous assignments of that atom type pair to sub−styles.

Note that you may need to use an atom_style hybrid command in your input script, if atoms in the simulation will need attributes from several atom styles, due to using multiple pair potentials.

The potential energy contribution to the overall system due to an individual sub−style can be accessed and output via the compute pair command.

IMPORTANT: Several of the potentials defined via the pair_style command in LAMMPS are really many−body potentials, such as Tersoff, AIREBO, MEAM, ReaxFF, etc. The way to think about using these potentials in a hybrid setting is as follows.

A subset of atom types is assigned to the many−body potential with a single pair_coeff command, using "* *" to include all types and the NULL keywords described above to exclude specific types not assigned to that potential. If types 1,3,4 were assigned in that way (but not type 2), this means that all many−body interactions between all atoms of types 1,3,4 will be computed by that potential. Pair_style hybrid allows interactions between type pairs 2−2, 1−2, 2−3, 2−4 to be specified for computation by other pair styles. You could even add a second interaction for 1−1 to be computed by another pair style, assuming pair_style hybrid/overlay is used.

But you should not, as a general rule, attempt to exclude the many−body interactions for some subset of the type pairs within the set of 1,3,4 interactions, e.g. exclude 1−1 or 1−3 interactions. That is not conceptually well−defined for many−body interactions, since the potential will typically calculate energies and foces for small groups of atoms, e.g. 3 or 4 atoms, using the neighbor lists of the atoms to find the additional atoms in the group. It is typically non−physical to think of excluding an interaction between a particular pair of atoms when the potential computes 3−body or 4−body interactions.

However, you can still use the pair_coeff none setting or the neigh_modify exclude command to exclude certain type pairs from the neighbor list that will be passed to a manybody sub−style. This will alter the calculations made by a many−body potential, since it builds its list of 3−body, 4−body, etc interactions from the pair list. You will need to think carefully as to whether it produces a physically meaningful result for your model.

For example, imagine you have two atom types in your model, type 1 for atoms in one surface, and type 2 for atoms in the other, and you wish to use a Tersoff potential to compute interactions within each surface, but not between surfaces. Then either of these two command sequences would implement that model:

```
pair_style hybrid tersoff
pair_coeff * * tersoff SiC.tersoff C C
pair_coeff 1 2 none

pair_style tersoff
pair_coeff * * SiC.tersoff C C
neigh_modify exclude type 1 2
```

Either way, only neighbor lists with 1−1 or 2−2 interactions would be passed to the Tersoff potential, which means it would compute no 3−body interactions containing both type 1 and 2 atoms.

Mixing, shift, table, tail correction, restart, rRESPA info:

Any pair potential settings made via the pair_modify command are passed along to all sub−styles of the hybrid potential.

For atom type pairs I,J and I != J, if the sub−style assigned to I,I and J,J is the same, and if the sub−style allows for mixing, then the coefficients for I,J can be mixed. This means you do not have to specify a pair_coeff command for I,J since the I,J type pair will be assigned automatically to the I,I sub−style and its coefficients generated by the mixing rule used by that sub−style. For the *hybrid/overlay* style, there is an additional

requirement that both the I,I and J,J pairs are assigned to a single sub−style. See the "pair_modify" command for details of mixing rules. See the See the doc page for the sub−style to see if allows for mixing.

The hybrid pair styles supports the pair_modify shift, table, and tail options for an I,J pair interaction, if the associated sub−style supports it.

For the hybrid pair styles, the list of sub−styles and their respective settings are written to binary restart files, so a pair_style command does not need to specified in an input script that reads a restart file. However, the coefficient information is not stored in the restart file. Thus, pair_coeff commands need to be re−specified in the restart input script.

These pair styles support the use of the *inner*, *middle*, and *outer* keywords of the run_style respa command, if their sub−styles do.

Restrictions:

When using a long−range Coulombic solver (via the kspace_style command) with a hybrid pair_style, one or more sub−styles will be of the "long" variety, e.g. *lj/cut/coul/long* or *buck/coul/long*. You must insure that the short−range Coulombic cutoff used by each of these long pair styles is the same or else LAMMPS will generate an error.

Related commands:

pair_coeff

Default: none

pair_style lj/cut command

pair_style lj/cut/cuda command

pair_style lj/cut/experimental/cuda command

pair_style lj/cut/gpu command

pair_style lj/cut/opt command

pair_style lj/cut/coul/cut command

pair_style lj/cut/coul/cut/cuda command

pair_style lj/cut/coul/cut/gpu command

pair_style lj/cut/coul/debye command

pair_style lj/cut/coul/debye/cuda command

pair_style lj/cut/coul/long command

pair_style lj/cut/coul/long/cuda command

pair_style lj/cut/coul/long/gpu command

pair_style lj/cut/coul/long/tip4p command

Syntax:

```
pair_style style args
```

- style = *lj/cut* or *lj/cut/coul/cut* or *lj/cut/coul/debye* or *lj/cut/coul/long* or *lj/cut/coul/long/tip4p*
- args = list of arguments for a particular style

```
lj/cut args = cutoff
    cutoff = global cutoff for Lennard Jones interactions (distance units)
  lj/cut/coul/cut args = cutoff (cutoff2)
    cutoff = global cutoff for LJ (and Coulombic if only 1 arg) (distance units)
    cutoff2 = global cutoff for Coulombic (optional) (distance units)
  lj/cut/coul/debye args = kappa cutoff (cutoff2)
    kappa = Debye length (inverse distance units)
    cutoff = global cutoff for LJ (and Coulombic if only 1 arg) (distance units)
    cutoff2 = global cutoff for Coulombic (optional) (distance units)
  lj/cut/coul/long args = cutoff (cutoff2)
    cutoff = global cutoff for LJ (and Coulombic if only 1 arg) (distance units)
    cutoff2 = global cutoff for Coulombic (optional) (distance units)
  lj/cut/coul/long/tip4p args = otype htype btype atype qdist cutoff (cutoff2)
    otype,htype = atom types for TIP4P O and H
```

```
    btype,atype = bond and angle types for TIP4P waters
    qdist = distance from O atom to massless charge (distance units)
    cutoff = global cutoff for LJ (and Coulombic if only 1 arg) (distance units)
    cutoff2 = global cutoff for Coulombic (optional) (distance units)
```

Examples:

```
pair_style lj/cut 2.5
pair_coeff * * 1 1
pair_coeff 1 1 1 1.1 2.8

pair_style lj/cut/coul/cut 10.0
pair_style lj/cut/coul/cut 10.0 8.0
pair_coeff * * 100.0 3.0
pair_coeff 1 1 100.0 3.5 9.0
pair_coeff 1 1 100.0 3.5 9.0 9.0

pair_style lj/cut/coul/debye 1.5 3.0
pair_style lj/cut/coul/debye 1.5 2.5 5.0
pair_coeff * * 1.0 1.0
pair_coeff 1 1 1.0 1.5 2.5
pair_coeff 1 1 1.0 1.5 2.5 5.0

pair_style lj/cut/coul/long 10.0
pair_style lj/cut/coul/long 10.0 8.0
pair_coeff * * 100.0 3.0
pair_coeff 1 1 100.0 3.5 9.0

pair_style lj/cut/coul/long/tip4p 1 2 7 8 0.3 12.0
pair_style lj/cut/coul/long/tip4p 1 2 7 8 0.3 12.0 10.0
pair_coeff * * 100.0 3.0
pair_coeff 1 1 100.0 3.5 9.0
```

Description:

The *lj/cut* styles compute the standard 12/6 Lennard−Jones potential, given by

$$E = 4\epsilon \left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right] \qquad r < r_c$$

Rc is the cutoff.

Style *lj/cut/coul/cut* adds a Coulombic pairwise interaction given by

$$E = \frac{C q_i q_j}{\epsilon r} \qquad r < r_c$$

where C is an energy−conversion constant, Qi and Qj are the charges on the 2 atoms, and epsilon is the dielectric constant which can be set by the dielectric command. If one cutoff is specified in the pair_style command, it is used for both the LJ and Coulombic terms. If two cutoffs are specified, they are used as cutoffs for the LJ and Coulombic terms respectively.

Style *lj/cut/coul/debye* adds an additional exp() damping factor to the Coulombic term, given by

$$E = \frac{C q_i q_j}{\epsilon r} \exp(-\kappa r) \qquad r < r_c$$

where kappa is the Debye length. This potential is another way to mimic the screening effect of a polar solvent.

Style *lj/cut/coul/long* computes the same Coulombic interactions as style *lj/cut/coul/cut* except that an additional damping factor is applied to the Coulombic term so it can be used in conjunction with the kspace_style command and its *ewald* or *pppm* option. The Coulombic cutoff specified for this style means that pairwise interactions within this distance are computed directly; interactions outside that distance are computed in reciprocal space.

Style *lj/cut/coul/long/tip4p* implements the TIP4P water model of (Jorgensen), which introduces a massless site located a short distance away from the oxygen atom along the bisector of the HOH angle. The atomic types of the oxygen and hydrogen atoms, the bond and angle types for OH and HOH interactions, and the distance to the massless charge site are specified as pair_style arguments.

IMPORTANT NOTE: For each TIP4P water molecule in your system, the atom IDs for the O and 2 H atoms must be consecutive, with the O atom first. This is to enable LAMMPS to "find" the 2 H atoms associated with each O atom. For example, if the atom ID of an O atom in a TIP4P water molecule is 500, then its 2 H atoms must have IDs 501 and 502.

See the howto section for more information on how to use the TIP4P pair style.

The following coefficients must be defined for each pair of atoms types via the pair_coeff command as in the examples above, or in the data file or restart files read by the read_data or read_restart commands, or by mixing as described below:

- epsilon (energy units)
- sigma (distance units)
- cutoff1 (distance units)
- cutoff2 (distance units)

Note that sigma is defined in the LJ formula as the zero−crossing distance for the potential, not as the energy minimum at 2^(1/6) sigma.

The latter 2 coefficients are optional. If not specified, the global LJ and Coulombic cutoffs specified in the pair_style command are used. If only one cutoff is specified, it is used as the cutoff for both LJ and Coulombic interactions for this type pair. If both coefficients are specified, they are used as the LJ and Coulombic cutoffs for this type pair. You cannot specify 2 cutoffs for style *lj/cut*, since it has no Coulombic terms.

For *lj/cut/coul/long* and *lj/cut/coul/long/tip4p* only the LJ cutoff can be specified since a Coulombic cutoff cannot be specified for an individual I,J type pair. All type pairs use the same global Coulombic cutoff specified in the pair_style command.

Styles with a *cuda*, *gpu*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in this section of the manual. The accelerated styles take the same arguments and should produce the same results, except for round−off and precision issues.

These accelerated styles are part of the "user−cuda", "gpu", and "opt" packages respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the −suffix command−line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See this section of the manual for more instructions on how to use the accelerated styles effectively.

Mixing, shift, table, tail correction, restart, rRESPA info:

For atom type pairs I,J and I != J, the epsilon and sigma coefficients and cutoff distance for all of the lj/cut pair styles can be mixed. The default mix value is *geometric*. See the "pair_modify" command for details.

All of the lj/cut pair styles support the pair_modify shift option for the energy of the Lennard−Jones portion of the pair interaction.

The *lj/cut/coul/long* and *lj/cut/coul/long/tip4p* pair styles support the pair_modify table option since they can tabulate the short−range portion of the long−range Coulombic interaction.

All of the lj/cut pair styles support the pair_modify tail option for adding a long−range tail correction to the energy and pressure for the Lennard−Jones portion of the pair interaction.

All of the lj/cut pair styles write their information to binary restart files, so pair_style and pair_coeff commands do not need to be specified in an input script that reads a restart file.

The lj/cut and lj/cut/coul/long pair styles support the use of the *inner*, *middle*, and *outer* keywords of the run_style respa command, meaning the pairwise forces can be partitioned by distance at different levels of the rRESPA hierarchy. The other styles only support the *pair* keyword of run_style respa. See the run_style command for details.

Restrictions:

The *lj/cut/coul/long* and *lj/cut/coul/long/tip4p* styles are part of the "kspace" package. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info. Note that the kspace package is installed by default.

Related commands:

pair_coeff

Default: none

(Jorgensen) Jorgensen, Chandrasekhar, Madura, Impey, Klein, J Chem Phys, 79, 926 (1983).

pair_style lj96/cut command

pair_style lj96/cut/cuda command

pair_style lj96/cut/gpu command

Syntax:

```
pair_style lj96/cut cutoff
```

- cutoff = global cutoff for lj96/cut interactions (distance units)

Examples:

```
pair_style lj96/cut 2.5
pair_coeff * * 1.0 1.0 4.0
pair_coeff 1 1 1.0 1.0
```

Description:

The *lj96/cut* style compute a 9/6 Lennard−Jones potential, instead of the standard 12/6 potential, given by

$$E = 4\epsilon \left[\left(\frac{\sigma}{r}\right)^9 - \left(\frac{\sigma}{r}\right)^6 \right] \qquad r < r_c$$

Rc is the cutoff.

The following coefficients must be defined for each pair of atoms types via the pair_coeff command as in the examples above, or in the data file or restart files read by the read_data or read_restart commands, or by mixing as described below:

- epsilon (energy units)
- sigma (distance units)
- cutoff (distance units)

The last coefficient is optional. If not specified, the global LJ cutoff specified in the pair_style command is used.

Styles with a *cuda*, *gpu*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in this section of the manual. The accelerated styles take the same arguments and should produce the same results, except for round−off and precision issues.

These accelerated styles are part of the "user−cuda", "gpu", and "opt" packages respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the −suffix command−line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See this section of the manual for more instructions on how to use the accelerated styles effectively.

Mixing, shift, table, tail correction, restart, rRESPA info:

For atom type pairs I,J and I != J, the epsilon and sigma coefficients and cutoff distance for all of the lj/cut pair styles can be mixed. The default mix value is *geometric*. See the "pair_modify" command for details.

This pair style supports the pair_modify shift option for the energy of the pair interaction.

The pair_modify table option is not relevant for this pair style.

This pair style supports the pair_modify tail option for adding a long−range tail correction to the energy and pressure of the pair interaction.

This pair style writes its information to binary restart files, so pair_style and pair_coeff commands do not need to be specified in an input script that reads a restart file.

This pair style supports the use of the *inner*, *middle*, and *outer* keywords of the run_style respa command, meaning the pairwise forces can be partitioned by distance at different levels of the rRESPA hierarchy. See the run_style command for details.

Restrictions: none

Related commands:

pair_coeff

Default: none

pair_style lj/coul command

Syntax:

```
pair_style lj/coul flag_lj flag_coul cutoff (cutoff2)
```

- flag_lj = *long* or *cut*

```
  long = use Kspace long-range summation for the dispersion term 1/r^6
    cut = use a cutoff
```

- flag_coul = *long* or *off*

```
  long = use Kspace long-range summation for the Coulombic term 1/r
    off = omit the Coulombic term
```

- cutoff = global cutoff for LJ (and Coulombic if only 1 cutoff) (distance units)
- cutoff2 = global cutoff for Coulombic (optional) (distance units)

Examples:

```
pair_style lj/coul cut off 2.5
pair_style lj/coul cut long 2.5 4.0
pair_style lj/coul long long 2.5 4.0
pair_coeff * * 1 1
pair_coeff 1 1 1 3 4
```

Description:

The *lj/coul* style computes the standard 12/6 Lennard−Jones and Coulombic potentials, given by

$$E = 4\epsilon \left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right] \qquad r < r_c$$

$$E = \frac{C q_i q_j}{\epsilon r} \qquad r < r_c$$

where C is an energy−conversion constant, Qi and Qj are the charges on the 2 atoms, epsilon is the dielectric constant which can be set by the dielectric command, and Rc is the cutoff. If one cutoff is specified in the pair_style command, it is used for both the LJ and Coulombic terms. If two cutoffs are specified, they are used as cutoffs for the LJ and Coulombic terms respectively.

The purpose of this pair style is to capture long−range interactions resulting from both attractive 1/r^6 Lennard−Jones and Coulombic 1/r interactions. This is done by use of the *flag_lj* and *flag_coul* settings. The In 't Veld paper has more details on when it is appropriate to include long−range 1/r^6 interactions, using this potential.

If *flag_lj* is set to *long*, no cutoff is used on the LJ 1/r^6 dispersion term. The long−range portion is calculated by using the kspace_style ewald/n command. The specified LJ cutoff then determines which portion of the LJ interactions are computed directly by the pair potential versus which part is computed in reciprocal space via the

Kspace style. If *flag_lj* is set to *cut*, the LJ interactions are simply cutoff, as with pair_style lj/cut.

If *flag_coul* is set to *long*, no cutoff is used on the Coulombic interactions. The long−range portion is calculated by using any style, including *ewald/n* of the kspace_style command. Note that if *flag_lj* is also set to long, then only the *ewald/n* Kspace style can perform the long−range calculations for both the LJ and Coulombic interactions. If *flag_coul* is set to *off*, Coulombic interactions are not computed.

The following coefficients must be defined for each pair of atoms types via the pair_coeff command as in the examples above, or in the data file or restart files read by the read_data or read_restart commands, or by mixing as described below:

- epsilon (energy units)
- sigma (distance units)
- cutoff1 (distance units)
- cutoff2 (distance units)

Note that sigma is defined in the LJ formula as the zero−crossing distance for the potential, not as the energy minimum at 2^(1/6) sigma.

The latter 2 coefficients are optional. If not specified, the global LJ and Coulombic cutoffs specified in the pair_style command are used. If only one cutoff is specified, it is used as the cutoff for both LJ and Coulombic interactions for this type pair. If both coefficients are specified, they are used as the LJ and Coulombic cutoffs for this type pair. Note that if you are using *flag_lj* set to *long*, you cannot specify a LJ cutoff for an atom type pair, since only one global LJ cutoff is allowed. Similarly, if you are using *flag_coul* set to *long*, you cannot specify a Coulombic cutoff for an atom type pair, since only one global Coulombic cutoff is allowed.

Mixing, shift, table, tail correction, restart, rRESPA info:

For atom type pairs I,J and I != J, the epsilon and sigma coefficients and cutoff distance for all of the lj/cut pair styles can be mixed. The default mix value is *geometric*. See the "pair_modify" command for details.

This pair style supports the pair_modify shift option for the energy of the Lennard−Jones portion of the pair interaction, assuming *flag_lj* is *cut*.

This pair style supports the pair_modify table option since it can tabulate the short−range portion of the long−range Coulombic interaction.

This pair style does not support the pair_modify tail option for adding a long−range tail correction to the Lennard−Jones portion of the energy and pressure.

This pair style writes its information to binary restart files, so pair_style and pair_coeff commands do not need to be specified in an input script that reads a restart file.

This pair style supports the use of the *inner*, *middle*, and *outer* keywords of the run_style respa command, meaning the pairwise forces can be partitioned by distance at different levels of the rRESPA hierarchy. See the run_style command for details.

Restrictions:

This style is part of the "user−ewaldn" package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

Related commands:

pair_coeff

Default: none

(In 't Veld) In 't Veld, Ismail, Grest, J Chem Phys (accepted) (2007).

pair_style lj/expand command

pair_style lj/expand/cuda command

pair_style lj/expand/gpu command

Syntax:

```
pair_style lj/expand cutoff
```

- cutoff = global cutoff for lj/expand interactions (distance units)

Examples:

```
pair_style lj/expand 2.5
pair_coeff * * 1.0 1.0 0.5
pair_coeff 1 1 1.0 1.0 −0.2 2.0
```

Description:

Style *lj/expand* computes a LJ interaction with a distance shifted by delta which can be useful when particles are of different sizes, since it is different that using different sigma values in a standard LJ formula:

$$E = 4\epsilon \left[\left(\frac{\sigma}{r - \Delta} \right)^{12} - \left(\frac{\sigma}{r - \Delta} \right)^{6} \right] \qquad r < r_c + \Delta$$

Rc is the cutoff which does not include the delta distance. I.e. the actual force cutoff is the sum of cutoff + delta.

The following coefficients must be defined for each pair of atoms types via the pair_coeff command as in the examples above, or in the data file or restart files read by the read_data or read_restart commands, or by mixing as described below:

- epsilon (energy units)
- sigma (distance units)
- delta (distance units)
- cutoff (distance units)

The delta values can be positive or negative. The last coefficient is optional. If not specified, the global LJ cutoff is used.

Styles with a *cuda*, *gpu*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in this section of the manual. The accelerated styles take the same arguments and should produce the same results, except for round−off and precision issues.

These accelerated styles are part of the "user−cuda", "gpu", and "opt" packages respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the −suffix command−line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See this section of the manual for more instructions on how to use the accelerated styles effectively.

Mixing, shift, table, tail correction, restart, rRESPA info:

For atom type pairs I,J and I != J, the epsilon, sigma, and shift coefficients and cutoff distance for this pair style can be mixed. Shift is always mixed via an *arithmetic* rule. The other coefficients are mixed according to the pair_modify mix value. The default mix value is *geometric*. See the "pair_modify" command for details.

This pair style supports the pair_modify shift option for the energy of the pair interaction.

The pair_modify table option is not relevant for this pair style.

This pair style supports the pair_modify tail option for adding a long−range tail correction to the energy and pressure of the pair interaction.

This pair style writes its information to binary restart files, so pair_style and pair_coeff commands do not need to be specified in an input script that reads a restart file.

This pair style can only be used via the *pair* keyword of the run_style respa command. It does not support the *inner*, *middle*, *outer* keywords.

Restrictions: none

Related commands:

pair_coeff

Default: none

pair_style lj/sf command

Syntax:

```
pair_style lj/sf cutoff
```

- cutoff = global cutoff for Lennard−Jones interactions (distance units)

Examples:

```
pair_style lj/sf 2.5
pair_coeff * * 1.0 1.0
pair_coeff 1 1 1.0 1.0 3.0
```

Description:

Style *lj/sf* computes a truncated and force−shifted LJ interaction (Shifted Force Lennard−Jones), so that both the potential and the force go continuously to zero at the cutoff (Toxvaerd):

$$
\begin{aligned}
F &= F_{\mathrm{LJ}}(r) - F_{\mathrm{LJ}}(r_c) \qquad r < r_c \\
E &= E_{\mathrm{LJ}}(r) - E_{\mathrm{LJ}}(r_c) + (r - r_c) F_{\mathrm{LJ}}(r_c) \qquad r < r_c \\
\text{with} \qquad E_{\mathrm{LJ}}(r) &= 4\epsilon \left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right] \qquad \text{and} \qquad F_{\mathrm{LJ}}(r) = -E'_{\mathrm{LJ}}(r)
\end{aligned}
$$

The following coefficients must be defined for each pair of atoms types via the pair_coeff command as in the examples above, or in the data file or restart files read by the read_data or read_restart commands, or by mixing as described below:

- epsilon (energy units)
- sigma (distance units)
- cutoff (distance units)

The last coefficient is optional. If not specified, the global LJ cutoff specified in the pair_style command is used.

Mixing, shift, table, tail correction, restart, rRESPA info:

For atom type pairs I,J and I != J, the epsilon and sigma coefficients and cutoff distance for this pair style can be mixed. Rin is a cutoff value and is mixed like the cutoff. The default mix value is *geometric*. See the "pair_modify" command for details.

The pair_modify shift option is not relevant for this pair style, since the pair interaction goes to 0.0 at the cutoff.

The pair_modify table option is not relevant for this pair style.

This pair style does not support the pair_modify tail option for adding long−range tail corrections to energy and pressure, since the energy of the pair interaction is smoothed to 0.0 at the cutoff.

This pair style writes its information to binary restart files, so pair_style and pair_coeff commands do not need to be specified in an input script that reads a restart file.

This pair style can only be used via the *pair* keyword of the run_style respa command. It does not support the *inner*, *middle*, *outer* keywords.

Restrictions:

This pair style is part of the "user−misc" package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

Related commands:

pair_coeff

Default: none

(Toxvaerd) Toxvaerd, Dyre, J Chem Phys, 134, 081102 (2011).

pair_style lj/smooth command

pair_style lj/smooth/cuda command

Syntax:

```
pair_style lj/smooth Rin Rc
```

- Rin = inner cutoff beyond which force smoothing will be applied (distance units)
- Rc = outer cutoff for lj/smooth interactions (distance units)

Examples:

```
pair_style lj/smooth 8.0 10.0
pair_coeff * * 10.0 1.5
pair_coeff 1 1 20.0 1.3 7.0 9.0
```

Description:

Style *lj/smooth* computes a LJ interaction with a force smoothing applied between the inner and outer cutoff.

$$
\begin{aligned}
E &= 4\epsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right] \qquad r < r_{in} \\
F &= C_1 + C_2(r - r_{in}) + C_3(r - r_{in})^2 + C_4(r - r_{in})^3 \qquad r_{in} < r < r_c
\end{aligned}
$$

The polynomial coefficients C1, C2, C3, C4 are computed by LAMMPS to cause the force to vary smoothly from the inner cutoff Rin to the outer cutoff Rc.

At the inner cutoff the force and its 1st derivative will match the unsmoothed LJ formula. At the outer cutoff the force and its 1st derivative will be 0.0. The inner cutoff cannot be 0.0.

IMPORTANT NOTE: this force smoothing causes the energy to be discontinuous both in its values and 1st derivative. This can lead to poor energy conservation and may require the use of a thermostat. Plot the energy and force resulting from this formula via the pair_write command to see the effect.

The following coefficients must be defined for each pair of atoms types via the pair_coeff command as in the examples above, or in the data file or restart files read by the read_data or read_restart commands, or by mixing as described below:

- epsilon (energy units)
- sigma (distance units)
- innner (distance units)
- outer (distance units)

The last 2 coefficients are optional inner and outer cutoffs. If not specified, the global values for Rin and Rc are used.

Styles with a *cuda*, *gpu*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in this section of the manual. The accelerated styles take the same arguments and should produce the same results, except for round−off and precision issues.

These accelerated styles are part of the "user−cuda", "gpu", and "opt" packages respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the −suffix command−line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See this section of the manual for more instructions on how to use the accelerated styles effectively.

Mixing, shift, table, tail correction, restart, rRESPA info:

For atom type pairs I,J and I != J, the epsilon, sigma, Rin coefficients and the cutoff distance for this pair style can be mixed. Rin is a cutoff value and is mixed like the cutoff. The other coefficients are mixed according to the pair_modify mix option. The default mix value is *geometric*. See the "pair_modify" command for details.

This pair style supports the pair_modify shift option for the energy of the pair interaction.

The pair_modify table option is not relevant for this pair style.

This pair style does not support the pair_modify tail option for adding long−range tail corrections to energy and pressure, since the energy of the pair interaction is smoothed to 0.0 at the cutoff.

This pair style writes its information to binary restart files, so pair_style and pair_coeff commands do not need to be specified in an input script that reads a restart file.

This pair style can only be used via the *pair* keyword of the run_style respa command. It does not support the *inner*, *middle*, *outer* keywords.

Restrictions: none

Related commands:

pair_coeff

Default: none

pair_style lubricate command

Syntax:

```
pair_style lubricate mu squeeze shear pump twist cutinner cutoff T_target seed
```

- mu = dynamic viscosity (dynamic viscosity units)
- squeeze = 0/1 for squeeze force off/on
- shear = 0/1 for shear force off/on
- pump = 0/1 for pump force off/on
- twist = 0/1 for twist force off/on
- cutinner = (distance units)
- cutoff = outer cutoff for interactions (distance units)
- T_target = desired temperature (temperature units)
- seed = random number seed (positive integer)

Examples:

```
pair_style lubricate 1.5 1 1 1 0 2.3 2.4 1.3 5878598
pair_coeff 1 1 1.8 2.0
pair_coeff * *

pair_style lubricate 1.0 1 1 1 0 2.3 2.4 1.3 5878598
pair_coeff * *
variable mu equal ramp(1,2)
fix 1 all adapt 1 pair lubricate mu * * v_mu
```

Description:

Style *lubricate* computes pairwise interactions between mono−disperse spherical particles via this formula from (Ball and Melrose)

$$W = -a_{sq}|(v_1 - v_2) \bullet \mathbf{nn}|^2 - a_{sh}|(\omega_1 + \omega_2) \bullet (\mathbf{I} - \mathbf{nn}) - 2\Omega_N|^2 - a_{pu}|(\omega_1 - \omega_2) \bullet (\mathbf{I} - \mathbf{nn})|^2 - a_{tw}|(\omega_1 - \omega_2) \bullet \mathbf{nn}|^2 \qquad r < r_c$$

$$\Omega_N = \mathbf{n} \times (v_1 - v_2)/r$$

which represents the dissipation W between two nearby particles due to their relative velocities in the presence of a background solvent with viscosity mu. Note that this is dynamic viscosity which has units of mass/distance/time, not kinematic viscosity.

The viscosity mu can be varied in a time−dependent manner over the course of a simluation, in which case in which case the pair_style setting for mu will be overridden. See the fix adapt command for details.

Rc is the outer cutoff specified in the pair_style command, the translational velocities of the 2 particles are v1 and v2, the angular velocities are w1 and w2, and n is the unit vector in the direction from particle 1 to 2. The 4 terms represent four modes of pairwise interaction: squeezing, shearing, pumping, and twisting. The 4 flags in the pair_style command turn on or off each of these modes by including or excluding each term. The 4 coefficients on each term are functions of the separation distance of the particles and the viscosity. Details are given in (Ball and

Melrose), including the forces and torques that result from taking derivatives of this equation with respect to velocity (see Appendix A).

Unlike most pair potentials, the two specified cutoffs (cutinner and cutoff) refer to the surface−to−surface separation between two particles, not center−to−center distance. Currently, this pair style can only be used for mono−disperse extended spheres (same radii), so that separation is r_ij − 2*radius, where r_ij is the center−to−center distance between the particles. Within the inner cutoff *cutinner*, the forces and torques are evaluated at a separation of cutinner. The outer *cutoff* is the separation distance beyond which the pair−wise forces are zero.

A Langevin thermostatting term is also added to the pairwise force, similar to that provided by the fix langevin or pair_style dpd commands. The target temperature for the thermostat is the specified *T_target*. The *seed* is used for the random numbers generated for the thermostat.

The following coefficients must be defined for each pair of atoms types via the pair_coeff command as in the examples above, or in the data file or restart files read by the read_data or read_restart commands, or by mixing as described below:

- cutinner (distance units)
- cutoff (distance units)

The two coefficients are optional. If neither is specified, the two cutoffs specified in the pair_style command are used. Otherwise both must be specified.

Mixing, shift, table, tail correction, restart, rRESPA info:

For atom type pairs I,J and I != J, the two cutoff distances for this pair style can be mixed. The default mix value is *geometric*. See the "pair_modify" command for details.

This pair style does not support the pair_modify shift option for the energy of the pair interaction.

The pair_modify table option is not relevant for this pair style.

This pair style does not support the pair_modify tail option for adding long−range tail corrections to energy and pressure.

This pair style writes its information to binary restart files, so pair_style and pair_coeff commands do not need to be specified in an input script that reads a restart file.

This pair style can only be used via the *pair* keyword of the run_style respa command. It does not support the *inner*, *middle*, *outer* keywords.

Restrictions:

This style is part of the "colloid" package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

This pair style requires that atoms be finite−size spheres with a diameter, as defined by the atom_style sphere command.

Per−particle or per−type polydispersity is not yet supported by this pair style; all particles must have the same diameter.

This pair style requires you to use the communicate vel yes option so that velocites are stored by ghost atoms.

Related commands:

pair_coeff

Default: none

(Ball) Ball and Melrose, Physica A, 247, 444−472 (1997).

pair_style meam command

Syntax:

```
pair_style meam
```

Examples:

```
pair_style meam
pair_coeff * * ../potentials/library.meam Si ../potentials/si.meam Si
pair_coeff * * ../potentials/library.meam Ni Al NULL Ni Al Ni Ni
```

Description:

NOTE: The behavior of the MEAM potential for alloy systems has changed as of November 2010; see description below of the mixture_ref_t parameter

Style *meam* computes pairwise interactions for a variety of materials using modified embedded−atom method (MEAM) potentials (Baskes). Conceptually, it is an extension to the original EAM potentials which adds angular forces. It is thus suitable for modeling metals and alloys with fcc, bcc, hcp and diamond cubic structures, as well as covalently bonded materials like silicon and carbon.

In the MEAM formulation, the total energy E of a system of atoms is given by:

$$E = \sum_i \left\{ F_i(\bar{\rho}_i) + \frac{1}{2} \sum_{i \neq j} \phi_{ij}(r_{ij}) \right\}$$

where F is the embedding energy which is a function of the atomic electron density rho, and phi is a pair potential interaction. The pair interaction is summed over all neighbors J of atom I within the cutoff distance. As with EAM, the multi−body nature of the MEAM potential is a result of the embedding energy term. Details of the computation of the embedding and pair energies, as implemented in LAMMPS, are given in (Gullet) and references therein.

The various parameters in the MEAM formulas are listed in two files which are specified by the pair_coeff command. These are ASCII text files in a format consistent with other MD codes that implement MEAM potentials, such as the serial DYNAMO code and Warp. Several MEAM potential files with parameters for different materials are included in the "potentials" directory of the LAMMPS distribution with a ".meam" suffix. All of these are parameterized in terms of LAMMPS metal units.

Note that unlike for other potentials, cutoffs for MEAM potentials are not set in the pair_style or pair_coeff command; they are specified in the MEAM potential files themselves.

Only a single pair_coeff command is used with the *meam* style which specifies two MEAM files and the element(s) to extract information for. The MEAM elements are mapped to LAMMPS atom types by specifying N additional arguments after the 2nd filename in the pair_coeff command, where N is the number of LAMMPS atom types:

- MEAM library file

- Elem1, Elem2, ...
- MEAM parameter file
- N element names = mapping of MEAM elements to atom types

As an example, the potentials/library.meam file has generic MEAM settings for a variety of elements. The potentials/sic.meam file has specific parameter settings for a Si and C alloy system. If your LAMMPS simulation has 4 atoms types and you want the 1st 3 to be Si, and the 4th to be C, you would use the following pair_coeff command:

```
pair_coeff * * library.meam Si C sic.meam Si Si Si C
```

The 1st 2 arguments must be * * so as to span all LAMMPS atom types. The two filenames are for the library and parameter file respectively. The Si and C arguments (between the file names) are the two elements for which info will be extracted from the library file. The first three trailing Si arguments map LAMMPS atom types 1,2,3 to the MEAM Si element. The final C argument maps LAMMPS atom type 4 to the MEAM C element.

If the 2nd filename is specified as NULL, no parameter file is read, which simply means the generic parameters in the library file are used. Use of the NULL specification for the parameter file is discouraged for systems with more than a single element type (e.g. alloys), since the parameter file is expected to set element interaction terms that are not captured by the information in the library file.

If a mapping value is specified as NULL, the mapping is not performed. This can be used when a *meam* potential is used as part of the *hybrid* pair style. The NULL values are placeholders for atom types that will be used with other potentials.

The MEAM library file provided with LAMMPS has the name potentials/library.meam. It is the "meamf" file used by other MD codes. Aside from blank and comment lines (start with #) which can appear anywhere, it is formatted as a series of entries, each of which has 19 parameters and can span multiple lines:

elt, lat, z, ielement, atwt, alpha, b0, b1, b2, b3, alat, esub, asub, t0, t1, t2, t3, rozero, ibar

The "elt" and "lat" parameters are text strings, such as elt = Si or Cu and lat = dia or fcc. Because the library file is used by Fortran MD codes, these strings may be enclosed in single quotes, but this is not required. The other numeric parameters match values in the formulas above. The value of the "elt" string is what is used in the pair_coeff command to identify which settings from the library file you wish to read in. There can be multiple entries in the library file with the same "elt" value; LAMMPS reads the 1st matching entry it finds and ignores the rest.

Other parameters in the MEAM library file correspond to single−element potential parameters:

```
lat      = lattice structure of reference configuration
z        = number of nearest neighbors in the reference structure
ielement = atomic number
atwt     = atomic weight
alat     = lattice constant of reference structure
esub     = energy per atom (eV) in the reference structure at equilibrium
asub     = "A" parameter for MEAM (see e.g. (Baskes))
```

The alpha, b0, b1, b2, b3, t0, t1, t2, t3 parameters correspond to the standard MEAM parameters in the literature (Baskes) (the b parameters are the standard beta parameters). The rozero parameter is an element−dependent density scaling that weights the reference background density (see e.g. equation 4.5 in (Gullet)) and is typically 1.0 for single−element systems. The ibar parameter selects the form of the function G(Gamma) used to compute the electron density; options are

```
  0 => G = sqrt(1+Gamma)
  1 => G = exp(Gamma/2)
  2 => not implemented
  3 => G = 2/(1+exp(-Gamma))
  4 => G = sqrt(1+Gamma)
 -5 => G = +-sqrt(abs(1+Gamma))
```

If used, the MEAM parameter file contains settings that override or complement the library file settings. Examples of such parameter files are in the potentials directory with a ".meam" suffix. Their format is the same as is read by other Fortran MD codes. Aside from blank and comment lines (start with #) which can appear anywhere, each line has one of the following forms. Each line can also have a trailing comment (starting with #) which is ignored.

```
keyword = value
keyword(I) = value
keyword(I,J) = value
keyword(I,J,K) = value
```

The recognized keywords are as follows:

Ec, alpha, rho0, delta, lattce, attrac, repuls, nn2, Cmin, Cmax, rc, delr, augt1, gsmooth_factor, re

where

```
rc          = cutoff radius for cutoff function; default = 4.0
delr        = length of smoothing distance for cutoff function; default = 0.1
rho0(I)     = relative density for element I (overwrites value
              read from meamf file)
Ec(I,J)     = cohesive energy of reference structure for I-J mixture
delta(I,J)  = heat of formation for I-J alloy; if Ec_IJ is input as
              zero, then LAMMPS sets Ec_IJ = (Ec_II + Ec_JJ)/2 - delta_IJ
alpha(I,J)  = alpha parameter for pair potential between I and J (can
              be computed from bulk modulus of reference structure
re(I,J)     = equilibrium distance between I and J in the reference
              structure
Cmax(I,J,K) = Cmax screening parameter when I-J pair is screened
              by K (I<=J); default = 2.8
Cmin(I,J,K) = Cmin screening parameter when I-J pair is screened
              by K (I<=J); default = 2.0
lattce(I,J) = lattice structure of I-J reference structure:
                dia = diamond (interlaced fcc for alloy)
                fcc = face centered cubic
                bcc = body centered cubic
                dim = dimer
                b1  = rock salt (NaCl structure)
                hcp = hexagonal close-packed
                c11 = MoSi2 structure
                l12 = Cu3Au structure (lower case L, followed by 12)
                b2  = CsCl structure (interpenetrating simple cubic)
nn2(I,J)    = turn on second-nearest neighbor MEAM formulation for
              I-J pair (see for example (Lee)).
                0 = second-nearest neighbor formulation off
                1 = second-nearest neighbor formulation on
                default = 0
attrac(I,J) = additional cubic attraction term in Rose energy I-J pair potential
                default = 0
repuls(I,J) = additional cubic repulsive term in Rose energy I-J pair potential
                default = 0
zbl(I,J)    = blend the MEAM I-J pair potential with the ZBL potential for small
              atom separations (ZBL)
                default = 1
```

```
gsmooth_factor  = factor determining the length of the G-function smoothing
                  region; only significant for ibar=0 or ibar=4.
                      99.0 = short smoothing region, sharp step
                      0.5  = long smoothing region, smooth step
                      default = 99.0
augt1           = integer flag for whether to augment t1 parameter by
                  3/5*t3 to account for old vs. new meam formulations;
                    0 = don't augment t1
                    1 = augment t1
                    default = 1
ialloy          = integer flag to use alternative averaging rule for t parameters,
                  for comparison with the DYNAMO MEAM code
                    0 = standard averaging (matches ialloy=0 in DYNAMO)
                    1 = alternative averaging (matches ialloy=1 in DYNAMO)
                    2 = no averaging of t (use single-element values)
                    default = 0
mixture_ref_t   = integer flag to use mixture average of t to compute the background
                  reference density for alloys, instead of the single-element values
                  (see description and warning elsewhere in this doc page)
                    0 = do not use mixture averaging for t in the reference density
                    1 = use mixture averaging for t in the reference density
                    default = 0
erose_form      = integer value to select the form of the Rose energy function
                  (see description below).
                    default = 0
emb_lin_neg     = integer value to select embedding function for negative densities
                    0 = F(rho)=0
                    1 = F(rho) = -asub*esub*rho (linear in rho, matches DYNAMO)
                    default = 0
bkgd_dyn        = integer value to select background density formula
                    0 = rho_bkgd = rho_ref_meam(a) (as in the reference structure)
                    1 = rho_bkgd = rho0_meam(a)*Z_meam(a) (matches DYNAMO)
                    default = 0
```

Rc, delr, re are in distance units (Angstroms in the case of metal units). Ec and delta are in energy units (eV in the case of metal units).

Each keyword represents a quantity which is either a scalar, vector, 2d array, or 3d array and must be specified with the correct corresponding array syntax. The indices I,J,K each run from 1 to N where N is the number of MEAM elements being used.

Thus these lines

```
rho0(2) = 2.25
alpha(1,2) = 4.37
```

set rho0 for the 2nd element to the value 2.25 and set alpha for the alloy interaction between elements 1 and 2 to 4.37.

The augt1 parameter is related to modifications in the MEAM formulation of the partial electron density function. In recent literature, an extra term is included in the expression for the third-order density in order to make the densities orthogonal (see for example (Wang), equation 3d); this term is included in the MEAM implementation in lammps. However, in earlier published work this term was not included when deriving parameters, including most of those provided in the library.meam file included with lammps, and to account for this difference the parameter t1 must be augmented by 3/5*t3. If augt1=1, the default, this augmentation is done automatically. When parameter values are fit using the modified density function, as in more recent literature, augt1 should be set to 0.

The mixture_ref_t parameter is available to match results with those of previous versions of lammps (before January 2011). Newer versions of lammps, by default, use the single−element values of the t parameters to compute the background reference density. This is the proper way to compute these parameters. Earlier versions of lammps used an alloy mixture averaged value of t to compute the background reference density. Setting mixture_ref_t=1 gives the old behavior. WARNING: using mixture_ref_t=1 will give results that are demonstrably incorrect for second−neighbor MEAM, and non−standard for first−neighbor MEAM; this option is included only for matching with previous versions of lammps and should be avoided if possible.

The parameters attrac and repuls, along with the integer selection parameter erose_form, can be used to modify the Rose energy function used to compute the pair potential. This function gives the energy of the reference state as a function of interatomic spacing. The form of this function is:

```
astar = alpha * (r/re − 1.d0)
if erose_form = 0: erose = −Ec*(1+astar+a3*(astar**3)/(r/re))*exp(−astar)
if erose_form = 1: erose = −Ec*(1+astar+(−attrac+repuls/r)*(astar**3))*exp(−astar)
if erose_form = 2: erose = −Ec*(1 +astar + a3*(astar**3))*exp(−astar)
a3 = repuls, astar <0
a3 = attrac, astar >= 0
```

Most published MEAM parameter sets use the default values attrac=repulse=0. Setting repuls=attrac=delta corresponds to the form used in several recent published MEAM parameter sets, such as (Vallone)

NOTE: The default form of the erose expression in LAMMPS was corrected in March 2009. The current version is correct, but may show different behavior compared with earlier versions of lammps with the attrac and/or repuls parameters are non−zero. To obtain the previous default form, use erose_form = 1 (this form does not seem to appear in the literature). An alternative form (see e.g. (Lee2)) is available using erose_form = 2.

Mixing, shift, table, tail correction, restart, rRESPA info:

For atom type pairs I,J and I != J, where types I and J correspond to two different element types, mixing is performed by LAMMPS with user−specifiable parameters as described above. You never need to specify a pair_coeff command with I != J arguments for this style.

This pair style does not support the pair_modify shift, table, and tail options.

This pair style does not write its information to binary restart files, since it is stored in potential files. Thus, you need to re−specify the pair_style and pair_coeff commands in an input script that reads a restart file.

This pair style can only be used via the *pair* keyword of the run_style respa command. It does not support the *inner*, *middle*, *outer* keywords.

Restrictions:

This style is part of the "meam" package. It is only enabled if LAMMPS was built with that package, which also requires the MEAM library be built and linked with LAMMPS. See the Making LAMMPS section for more info.

Related commands:

pair_coeff, pair_style eam

Default: none

(Baskes) Baskes, Phys Rev B, 46, 2727−2742 (1992).

(Gullet) Gullet, Wagner, Slepoy, SANDIA Report 2003−8782 (2003). This report may be accessed on−line via this link.

(Lee) Lee, Baskes, Phys. Rev. B, 62, 8564−8567 (2000).

(Lee2) Lee, Baskes, Kim, Cho. Phys. Rev. B, 64, 184102 (2001).

(Valone) Valone, Baskes, Martin, Phys. Rev. B, 73, 214209 (2006).

(Wang) Wang, Van Hove, Ross, Baskes, J. Chem. Phys., 121, 5410 (2004).

(ZBL) J.F. Ziegler, J.P. Biersack, U. Littmark, 'Stopping and Ranges of Ions in Matter' Vol 1, 1985, Pergamon Press.

pair_modify command

Syntax:

```
pair_modify keyword value ...
```

- one or more keyword/value pairs may be listed
- keyword = *shift* or *mix* or *table* or *tabinner* or *tail*

```
mix value = geometric or arithmetic or sixthpower
  shift value = yes or no
  table value = N
    2^N = # of values in table
  tabinner value = cutoff
    cutoff = inner cutoff at which to begin table (distance units)
  tail value = yes or no
```

Examples:

```
pair_modify shift yes mix geometric
pair_modify tail yes
pair_modify table 12
```

Description:

Modify the parameters of the currently defined pair style. Not all parameters are relevant to all pair styles.

The *mix* keyword affects pair coefficients for interactions between atoms of type I and J, when I != J and the coefficients are not explicitly set in the input script. Note that coefficients for I = J must be set explicitly, either in the input script via the "pair_coeff" command or in the "Pair Coeffs" section of the data file. For some pair styles it is not necessary to specify coefficients when I != J, since a "mixing" rule will create them from the I,I and J,J settings. The pair_modify *mix* value determines what formulas are used to compute the mixed coefficients. In each case, the cutoff distance is mixed the same way as sigma.

Note that not all pair styles support mixing. Also, some mix options are not available for certain pair styles. See the doc page for individual pair styles for those restrictions. Note also that the pair_coeff command also can be to directly set coefficients for a specific I != J pairing, in which case no mixing is performed.

mix *geometric*

```
epsilon_ij = sqrt(epsilon_i * epsilon_j)
sigma_ij = sqrt(sigma_i * sigma_j)
```

mix *arithmetic*

```
epsilon_ij = sqrt(epsilon_i * epsilon_j)
sigma_ij = (sigma_i + sigma_j) / 2
```

mix *sixthpower*

```
epsilon_ij = (2 * sqrt(epsilon_i*epsilon_j) * sigma_i^3 * sigma_j^3) /
             (sigma_i^6 + sigma_j^6)
sigma_ij = ((sigma_i**6 + sigma_j**6) / 2) ^ (1/6)
```

The *shift* keyword determines whether a Lennard−Jones potential is shifted at its cutoff to 0.0. If so, this adds an energy term to each pairwise interaction which will be included in the thermodynamic output, but does not affect pair forces or atom trajectories. See the doc page for individual pair styles to see which ones support this option.

The *table* keyword applies to pair styles with a long−range Coulombic term; see the doc page for individual styles to see which potentials support this option. If N is non−zero, a table of length 2^N is pre−computed for forces and energies, which can shrink their computational cost by up to a factor of 2. The table is indexed via a bit−mapping technique (Wolff) and a linear interpolation is performed between adjacent table values. In our experiments with different table styles (lookup, linear, spline), this method typically gave the best performance in terms of speed and accuracy.

The choice of table length is a tradeoff in accuracy versus speed. A larger N yields more accurate force computations, but requires more memory which can slow down the computation due to cache misses. A reasonable value of N is between 8 and 16. The default value of 12 (table of length 4096) gives approximately the same accuracy as the no−table (N = 0) option. For N = 0, forces and energies are computed directly, using a polynomial fit for the needed erfc() function evaluation, which is what earlier versions of LAMMPS did. Values greater than 16 typically slow down the simulation and will not improve accuracy; values from 1 to 8 give unreliable results.

The *tabinner* keyword sets an inner cutoff above which the pairwise computation is done by table lookup (if tables are invoked). The smaller this value is set, the less accurate the table becomes (for a given number of table values), which can require use of larger tables. The default cutoff value is sqrt(2.0) distance units which means nearly all pairwise interactions are computed via table lookup for simulations with "real" units, but some close pairs may be computed directly (non−table) for simulations with "lj" units.

When the *tail* keyword is set to *yes*, certain pair styles will add a long−range VanderWaals tail "correction" to the energy and pressure. See the doc page for individual styles to see which support this option. These corrections are included in the calculation and printing of thermodynamic quantities (see the thermo_style command). Their effect will also be included in constant NPT or NPH simulations where the pressure influences the simulation box dimensions (e.g. the fix npt and fix nph commands). The formulas used for the long−range corrections come from equation 5 of (Sun).

Several assumptions are inherent in using tail corrections, including the following:

- The simulated system is a 3d bulk homogeneous liquid. This option should not be used for systems that are non−liquid, 2d, have a slab geometry (only 2d periodic), or inhomogeneous.
- G(r), the radial distribution function (rdf), is unity beyond the cutoff, so a fairly large cutoff should be used (i.e. 2.5 sigma for an LJ fluid), and it is probably a good idea to verify this assumption by checking the rdf. The rdf is not exactly unity beyond the cutoff for each pair of interaction types, so the tail correction is necessarily an approximation.
- Thermophysical properties obtained from calculations with this option enabled will not be thermodynamically consistent with the truncated force−field that was used. In other words, atoms do not feel any LJ pair interactions beyond the cutoff, but the energy and pressure reported by the simulation include an estimated contribution from those interactions.

Restrictions: none

You cannot use *shift* yes with *tail* yes, since those are conflicting options. You cannot use *tail* yes with 2d simulations.

Related commands:

pair_style, pair_coeff, thermo_style

Default:

The option defaults are mix = geometric, shift = no, table = 12, tabinner = sqrt(2.0), tail = no.

Note that some pair styles perform mixing, but only a certain style of mixing. See the doc pages for individual pair styles for details.

(Wolff) Wolff and Rudd, Comp Phys Comm, 120, 200−32 (1999).

(Sun) Sun, J Phys Chem B, 102, 7338−7364 (1998).

pair_style morse command

pair_style morse/cuda command

pair_style morse/gpu command

pair_style morse/opt command

Syntax:

```
pair_style morse cutoff
```

- cutoff = global cutoff for Morse interactions (distance units)

Examples:

```
pair_style morse 2.5
pair_coeff * * 100.0 2.0 1.5
pair_coeff 1 1 100.0 2.0 1.5 3.0
```

Description:

Style *morse* computes pairwise interactions with the formula

$$E = D_0 \left[e^{-2\alpha(r-r_0)} - 2e^{-\alpha(r-r_0)} \right] \qquad r < r_c$$

Rc is the cutoff.

The following coefficients must be defined for each pair of atoms types via the pair_coeff command as in the examples above, or in the data file or restart files read by the read_data or read_restart commands:

- D0 (energy units)
- alpha (1/distance units)
- r0 (distance units)
- cutoff (distance units)

The last coefficient is optional. If not specified, the global morse cutoff is used.

Styles with a *cuda*, *gpu*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in this section of the manual. The accelerated styles take the same arguments and should produce the same results, except for round−off and precision issues.

These accelerated styles are part of the "user−cuda", "gpu", and "opt" packages respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the −suffix command−line switch when you invoke LAMMPS, or you can use the suffix command in your input

script.

See this section of the manual for more instructions on how to use the accelerated styles effectively.

Mixing, shift, table, tail correction, restart, rRESPA info:

None of these pair styles support mixing. Thus, coefficients for all I,J pairs must be specified explicitly.

All of these pair styles support the pair_modify shift option for the energy of the pair interaction.

The pair_modify table options is not relevant for the Morse pair styles.

None of these pair styles support the pair_modify tail option for adding long−range tail corrections to energy and pressure.

All of these pair styles write their information to binary restart files, so pair_style and pair_coeff commands do not need to be specified in an input script that reads a restart file.

These pair styles can only be used via the *pair* keyword of the run_style respa command. They do not support the *inner*, *middle*, *outer* keywords.

Restrictions: none

Related commands:

pair_coeff

Default: none

pair_style none command

Syntax:

```
pair_style none
```

Examples:

```
pair_style none
```

Description:

Using a pair style of none means pair forces are not computed.

With this choice, the force cutoff is 0.0, which means that only atoms within the neighbor skin distance (see the neighbor command) are communicated between processors. You must insure the skin distance is large enough to acquire atoms needed for computing bonds, angles, etc.

A pair style of *none* will also prevent pairwise neighbor lists from being built. However if the neighbor style is *bin*, data structures for binning are still allocated. If the neighbor skin distance is small, then these data structures can consume a large amount of memory. So you should either set the neighbor style to *nsq* or set the skin distance to a larger value.

Restrictions: none

Related commands: none

Default: none

pair_style peri/pmb command

pair_style peri/lps command

Syntax:

```
pair_style style
```

- style = *peri/pmb* or *peri/lps*

Examples:

```
pair_style peri/pmb
pair_coeff * * 1.6863e22 0.0015001 0.0005 0.25

pair_style peri/lps
pair_coeff * * 14.9e9 14.9e9 0.0015001 0.0005 0.25
```

Description:

The peridynamic pair styles implement material models that can be used at the mescscopic and macroscopic scales.

Style *peri/pmb* implements the Peridynamic bond−based prototype microelastic brittle (PMB) model.

Style *peri/lps* implements the Peridynamic state−based linear peridynamic solid (LPS) model.

The canonical papers on Peridynamics are (Silling 2000) and (Silling 2007). The implementation of Peridynamics in LAMMPS is described in (Parks). Also see the PDLAMMPS user guide for more details about the implementation of peridynamics in LAMMPS.

The following coefficients must be defined for each pair of atom types via the pair_coeff command as in the examples above, or in the data file or restart files read by the read_data or read_restart commands, or by mixing as described below.

For the *peri/pmb* style:

- c (energy/distance/volume^2 units)
- horizon (distance units)
- s00 (unitless)
- alpha (unitless)

C is the effectively a spring constant for Peridynamic bonds, the horizon is a cutoff distance for truncating interactions, and s00 and alpha are used as a bond breaking criteria. The units of c are such that c/distance = stiffness/volume^2, where stiffness is energy/distance^2 and volume is distance^3. See the users guide for more details.

For the *peri/lps* style:

- K (force/area units)
- G (force/area units)

- horizon (distance units)
- s00 (unitless)
- alpha (unitless)

K is the bulk modulus and G is the shear modulus. The horizon is a cutoff distance for truncating interactions, and s00 and alpha are used as a bond breaking criteria. See the users guide for more details.

Mixing, shift, table, tail correction, restart, rRESPA info:

These pair styles do not support mixing. Thus, coefficients for all I,J pairs must be specified explicitly.

These pair styles do not support the pair_modify shift option.

The pair_modify table and tail options are not relevant for these pair styles.

These pair styles write their information to binary restart files, so pair_style and pair_coeff commands do not need to be specified in an input script that reads a restart file.

These pair styles can only be used via the *pair* keyword of the run_style respa command. They do not support the *inner*, *middle*, *outer* keywords.

Restrictions:

The *peri/pmb* and *peri/lps* styles are part of the "peri" package. They are only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

Related commands:

pair_coeff

Default: none

(Parks) Parks, Lehoucq, Plimpton, Silling, Comp Phys Comm, 179(11), 777−783 (2008).

(Silling 2000) Silling, J Mech Phys Solids, 48, 175−209 (2000).

(Silling 2007) Silling, Epton, Weckner, Xu, Askari, J Elasticity, 88, 151−184 (2007).

pair_style reax command

Syntax:

```
pair_style reax hbcut hbnewflag tripflag precision
```

- hbcut = hydrogen−bond cutoff (optional) (distance units)
- hbnewflag = use old or new hbond function style (0 or 1) (optional)
- tripflag = apply stabilization to all triple bonds (0 or 1) (optional)
- precision = precision for charge equilibration (optional)

Examples:

```
pair_style reax
pair_style reax 10.0 0 1 1.0e−5
pair_coeff * * ffield.reax 3 1 2 2
pair_coeff * * ffield.reax 3 NULL NULL 3
```

Description:

Style *reax* computes the ReaxFF potential of van Duin, Goddard and co−workers. ReaxFF uses distance−dependent bond−order functions to represent the contributions of chemical bonding to the potential energy. There is more than one version of ReaxFF. The version implemented in LAMMPS uses the functional forms documented in the supplemental information of the following paper: (Chenoweth). The version integrated into LAMMPS matches the most up−to−date version of ReaxFF as of summer 2010.

The *reax* style differs from the pair_style reax/c command in the lo−level implementation details. The *reax* style is a Fortran library, linked to LAMMPS. The *reax/c* style was initially implemented as stand−alone C code and is now integrated into LAMMPS as a package.

LAMMPS requires that a file called ffield.reax be provided, containing the ReaxFF parameters for each atom type, bond type, etc. The format is identical to the ffield file used by van Duin and co−workers. The filename is required as an argument in the pair_coeff command. Any value other than "ffield.reax" will be rejected (see below).

LAMMPS provides several different versions of ffield.reax in its potentials dir, each called potentials/ffield.reax.label. These are documented in potentials/README.reax. The default ffield.reax contains parameterizations for the following elements: C, H, O, N, S.

The format of these files is identical to that used originally by van Duin. We have tested the accuracy of *pair_style reax* potential against the original ReaxFF code for the systems mentioned above. You can use other ffield files for specific chemical systems that may be available elsewhere (but note that their accuracy may not have been tested).

The *hbcut*, *hbnewflag*, *tripflag*, and *precision* settings are optional arguments. If none are provided, default settings are used: *hbcut* = 6 (which is Angstroms in real units), *hbnewflag* = 1 (use new hbond function style), *tripflag* = 1 (apply stabilization to all triple bonds), and *precision* = 1.0e−6 (one part in 10^6). If you wish to override any of these defaults, then all of the settings must be specified.

Two examples using *pair_style reax* are provided in the examples/reax sub−directory, along with corresponding examples for pair_style reax/c.

Use of this pair style requires that a charge be defined for every atom since the *reax* pair style performs a charge equilibration (QEq) calculation. See the atom_style and read_data commands for details on how to specify charges.

The thermo variable *evdwl* stores the sum of all the ReaxFF potential energy contributions, with the exception of the Coulombic and charge equilibration contributions which are stored in the thermo variable *ecoul*. The output of these quantities is controlled by the thermo command.

This pair style tallies a breakdown of the total ReaxFF potential energy into sub−categories, which can be accessed via the compute pair command as a vector of values of length 14. The 14 values correspond to the following sub−categories (the variable names in italics match those used in the ReaxFF FORTRAN library):

1. *eb* = bond energy
2. *ea* = atom energy
3. *elp* = lone−pair energy
4. *emol* = molecule energy (always 0.0)
5. *ev* = valence angle energy
6. *epen* = double−bond valence angle penalty
7. *ecoa* = valence angle conjugation energy
8. *ehb* = hydrogen bond energy
9. *et* = torsion energy
10. *eco* = conjugation energy
11. *ew* = van der Waals energy
12. *ep* = Coulomb energy
13. *efi* = electric field energy (always 0.0)
14. *eqeq* = charge equilibration energy

To print these quantities to the log file (with descriptive column headings) the following commands could be included in an input script:

```
compute reax all pair reax
variable eb      equal c_reax[1]
variable ea      equal c_reax[2]
...
variable eqeq    equal c_reax[14]
thermo_style custom step temp epair v_eb v_ea ... v_eqeq
```

Only a single pair_coeff command is used with the *reax* style which specifies a ReaxFF potential file with parameters for all needed elements. These are mapped to LAMMPS atom types by specifying N additional arguments after the filename in the pair_coeff command, where N is the number of LAMMPS atom types:

- filename
- N indices = mapping of ReaxFF elements to atom types

The specification of the filename and the mapping of LAMMPS atom types recognized by the ReaxFF is done differently than for other LAMMPS potentials, due to the non−portable difficulty of passing character strings (e.g. filename, element names) between C++ and Fortran.

The filename has to be "ffield.reax" and it has to exist in the directory you are running LAMMPS in. This means you cannot prepend a path to the file in the potentials dir. Rather, you should copy that file into the directory you are running from. If you wish to use another ReaxFF potential file, then name it "ffield.reax" and put it in the directory you run from.

In the ReaxFF potential file, near the top, after the general parameters, is the atomic parameters section that contains element names, each with a couple dozen numeric parameters. If there are M elements specified in the *ffield* file, think of these as numbered 1 to M. Each of the N indices you specify for the N atom types of LAMMPS atoms must be an integer from 1 to M. Atoms with LAMMPS type 1 will be mapped to whatever element you specify as the first index value, etc. If a mapping value is specified as NULL, the mapping is not performed. This can be used when a ReaxFF potential is used as part of the *hybrid* pair style. The NULL values are placeholders for atom types that will be used with other potentials.

IMPORTANT NOTE: Currently the reax pair style cannot be used as part of the *hybrid* pair style. Some additional changes still need to be made to enable this.

As an example, say your LAMMPS simulation has 4 atom types and the elements are ordered as C, H, O, N in the *ffield* file. If you want the LAMMPS atom type 1 and 2 to be C, type 3 to be N, and type 4 to be H, you would use the following pair_coeff command:

```
pair_coeff * * ffield.reax 1 1 4 2
```

Mixing, shift, table, tail correction, restart, rRESPA info:

This pair style does not support the pair_modify mix, shift, table, and tail options.

This pair style does not write its information to binary restart files, since it is stored in potential files. Thus, you need to re−specify the pair_style and pair_coeff commands in an input script that reads a restart file.

This pair style can only be used via the *pair* keyword of the run_style respa command. It does not support the *inner*, *middle*, *outer* keywords.

Restrictions:

The ReaxFF potential files provided with LAMMPS in the potentials directory are parameterized for real units. You can use the ReaxFF potential with any LAMMPS units, but you would need to create your own potential file with coefficients listed in the appropriate units if your simulation doesn't use "real" units.

Related commands:

pair_coeff, pair_style reax/c, fix_reax_bonds

Default:

The keyword defaults are *hbcut* = 6, *hbnewflag* = 1, *tripflag* = 1, *precision* = 1.0e−6.

(Chenoweth_2008) Chenoweth, van Duin and Goddard, Journal of Physical Chemistry A, 112, 1040−1053 (2008).

pair_style reax/c command

Syntax:

```
pair_style reax/c cfile keyword value
```

- cfile = NULL or name of a control file

```
one keyword value pair may be appended
keyword = checkqeq
  checkqeq value = yes or no = whether or not to require qeq/reax fix
```

Examples:

```
pair_style reax/c NULL
pair_style reax/c controlfile checkqeq no
pair_coeff * * ffield.reax 1 2 2 3
```

Description:

Style *reax/c* computes the ReaxFF potential of van Duin, Goddard and co−workers. ReaxFF uses distance−dependent bond−order functions to represent the contributions of chemical bonding to the potential energy. There is more than one version of ReaxFF. The version implemented in LAMMPS uses the functional forms documented in the supplemental information of the following paper: (Chenoweth et al., 2008). The version integrated into LAMMPS matches the most up−to−date version of ReaxFF as of summer 2010. For more technical details about the pair reax/c implementation of ReaxFF, see the "(Aktulga)" paper.

The *reax/c* style differs from the pair_style reax command in the lo−level implementation details. The *reax* style is a Fortran library, linked to LAMMPS. The *reax/c* style was initially implemented as stand−alone C code and is now integrated into LAMMPS as a package.

LAMMPS provides several different versions of ffield.reax in its potentials dir, each called potentials/ffield.reax.label. These are documented in potentials/README.reax. The default ffield.reax contains parameterizations for the following elements: C, H, O, N, S.

The format of these files is identical to that used originally by van Duin. We have tested the accuracy of *pair_style reax/c* potential against the original ReaxFF code for the systems mentioned above. You can use other ffield files for specific chemical systems that may be available elsewhere (but note that their accuracy may not have been tested).

The *cfile* setting can be specified as NULL, in which case default settings are used. Or a control file can be specified which contains cutoff values for the ReaxFF potential in addition to some performance and output controls. Each line in the control specifies the value for a control variable. The format of the control file is described below.

Two examples using *pair_style reax/c* are provided in the examples/reax sub−directory, along with corresponding examples for pair_style reax.

Use of this pair style requires that a charge be defined for every atom. See the atom_style and read_data commands for details on how to specify charges.

The ReaxFF parameter files provided were created using a charge equilibration (QEq) model for handling the electrostatic interactions. Therefore, by default, LAMMPS requires that the fix qeq/reax command be used with *pair_style reax/c* when simulating a ReaxFF model, to equilibrate charge each timestep. Using the keyword *checkqeq* with the value *no* turns off the check for *fix qeq/reax*, allowing a simulation to be run without charge equilibration. In this case, the static charges you assign to each atom will be used for computing the electrostatic interactions in the system. See the fix qeq/reax command for details.

The thermo variable *evdwl* stores the sum of all the ReaxFF potential energy contributions, with the exception of the Coulombic and charge equilibration contributions which are stored in the thermo variable *ecoul*. The output of these quantities is controlled by the thermo command.

This pair style tallies a breakdown of the total ReaxFF potential energy into sub−categories, which can be accessed via the compute pair command as a vector of values of length 14. The 14 values correspond to the following sub−categories (the variable names in italics match those used in the original FORTRAN ReaxFF code):

1. *eb* = bond energy
2. *ea* = atom energy
3. *elp* = lone−pair energy
4. *emol* = molecule energy (always 0.0)
5. *ev* = valence angle energy
6. *epen* = double−bond valence angle penalty
7. *ecoa* = valence angle conjugation energy
8. *ehb* = hydrogen bond energy
9. *et* = torsion energy
10. *eco* = conjugation energy
11. *ew* = van der Waals energy
12. *ep* = Coulomb energy
13. *efi* = electric field energy (always 0.0)
14. *eqeq* = charge equilibration energy

To print these quantities to the log file (with descriptive column headings) the following commands could be included in an input script:

```
compute reax all pair reax/c
variable eb      equal c_reax[1]
variable ea      equal c_reax[2]
...
variable eqeq    equal c_reax[14]
thermo_style custom step temp epair v_eb v_ea ... v_eqeq
```

Only a single pair_coeff command is used with the *reax/c* style which specifies a ReaxFF potential file with parameters for all needed elements. These are mapped to LAMMPS atom types by specifying N additional arguments after the filename in the pair_coeff command, where N is the number of LAMMPS atom types:

- filename
- N indices = mapping of ReaxFF elements to atom types

The filename is the ReaxFF potential file. Unlike for the *reax* pair style, any filename can be used.

In the ReaxFF potential file, near the top, after the general parameters, is the atomic parameters section that contains element names, each with a couple dozen numeric parameters. If there are M elements specified in the *ffield* file, think of these as numbered 1 to M. Each of the N indices you specify for the N atom types of

LAMMPS atoms must be an integer from 1 to M. Atoms with LAMMPS type 1 will be mapped to whatever element you specify as the first index value, etc. If a mapping value is specified as NULL, the mapping is not performed. This can be used when the *reax/c* style is used as part of the *hybrid* pair style. The NULL values are placeholders for atom types that will be used with other potentials.

IMPORTANT NOTE: Currently the reax/c pair style cannot be used as part of the *hybrid* pair style. Some additional work still need to be done to enable this.

As an example, say your LAMMPS simulation has 4 atom types and the elements are ordered as C, H, O, N in the *ffield* file. If you want the LAMMPS atom type 1 and 2 to be C, type 3 to be N, and type 4 to be H, you would use the following pair_coeff command:

```
pair_coeff * * ffield.reax 1 1 4 2
```

The format of a line in the control file is as follows:

```
variable_name value
```

and it may be followed by an "!" character and a trailing comment.

If the value of a control variable is not specified, then default values are used. What follows is the list of variables along with a brief description of their use and default values.

simulation_name: Output files produced by *pair_style reax/c* carry this name + extensions specific to their contents. Partial energies are reported with a ".pot" extension, while the trajectory file has ".trj" extension.

tabulate_long_range: To improve performance, long range interactions can optionally be tabulated (0 means no tabulation). Value of this variable denotes the size of the long range interaction table. The range from 0 to long range cutoff (defined in the *ffield* file) is divided into *tabulate_long_range* points. Then at the start of simulation, we fill in the entries of the long range interaction table by computing the energies and forces resulting from van der Waals and Coulomb interactions between every possible atom type pairs present in the input system. During the simulation we consult to the long range interaction table to estimate the energy and forces between a pair of atoms. Linear interpolation is used for estimation. (default value = 0)

energy_update_freq: Denotes the frequency (in number of steps) of writes into the partial energies file. (default value = 0)

nbrhood_cutoff: Denotes the near neighbors cutoff (in Angstroms) regarding the bonded interactions. (default value = 4)

hbond_cutoff: Denotes the cutoff distance (in Angstroms) for hydrogen bond interactions.(default value = 0 − means no hydrogen bonds are present)

bond_graph_cutoff: is the threshold used in determining what is a physical bond, what is not. Bonds and angles reported in the trajectory file rely on this cutoff. (default value = 0.3)

thb_cutoff: cutoff value for the strength of bonds to be considered in three body interactions. (default value = 0.001)

write_freq: Frequency of writes into the trajectory file. (default value = 0)

traj_title: Title of the trajectory − not the name of the trajectory file.

atom_info: 1 means print only atomic positions + charge (default = 0)

atom_forces: 1 adds net forces to atom lines in the trajectory file (default = 0)

atom_velocities: 1 adds atomic velocities to atoms line (default = 0)

bond_info: 1 prints bonds in the trajectory file (default = 0)

angle_info: 1 prints angles in the trajectory file (default = 0)

Mixing, shift, table, tail correction, restart, rRESPA info:

This pair style does not support the pair_modify mix, shift, table, and tail options.

This pair style does not write its information to binary restart files, since it is stored in potential files. Thus, you need to re−specify the pair_style and pair_coeff commands in an input script that reads a restart file.

This pair style can only be used via the *pair* keyword of the run_style respa command. It does not support the *inner*, *middle*, *outer* keywords.

Restrictions:

This pair style is part of the "user−reaxc" package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

The ReaxFF potential files provided with LAMMPS in the potentials directory are parameterized for real units. You can use the ReaxFF potential with any LAMMPS units, but you would need to create your own potential file with coefficients listed in the appropriate units if your simulation doesn't use "real" units.

This pair style cannot yet compute per−atom energy or stress. If you use another command that tries to calculate these quantities using this pair style, a warning message will be printed and the quantities will be 0.0.

Related commands:

pair_coeff, fix_qeq_reax, pair_style reax

Default:

The keyword default is checkqeq = yes.

(Chenoweth_2008) Chenoweth, van Duin and Goddard, Journal of Physical Chemistry A, 112, 1040−1053 (2008).

:link(Aktulga) **(Aktulga)** Aktulga, Fogarty, Pandit, Grama, Parallel Computing, to appear (2011).

pair_style resquared command

pair_style resquared/gpu command

Syntax:

```
pair_style resquared cutoff
```

- cutoff = global cutoff for interactions (distance units)

Examples:

```
pair_style resquared 10.0
pair_coeff * * 1.0 1.0 1.7 3.4 3.4 1.0 1.0 1.0
```

Description:

Style *resquared* computes the RE−squared anisotropic interaction (Everaers), (Babadi) between pairs of ellipsoidal and/or spherical Lennard−Jones particles. For ellipsoidal interactions, the potential considers the ellipsoid as being comprised of small spheres of size sigma. LJ particles are a single sphere of size sigma. The distinction is made to allow the pair style to make efficient calculations of ellipsoid/solvent interactions.

Details for the equations used are given in the references below and in this supplementary document.

Use of this pair style requires the NVE, NVT, or NPT fixes with the *asphere* extension (e.g. fix nve/asphere) in order to integrate particle rotation. Additionally, atom_style ellipsoid should be used since it defines the rotational state and the size and shape of each ellipsoidal particle.

The following coefficients must be defined for each pair of atoms types via the pair_coeff command as in the examples above, or in the data file or restart files read by the read_data or read_restart commands:

- A12 = Energy Prefactor/Hamaker constant (energy units)
- sigma = atomic interaction diameter (distance units)
- epsilon_i_a = relative well depth of type I for side−to−side interactions
- epsilon_i_b = relative well depth of type I for face−to−face interactions
- epsilon_i_c = relative well depth of type I for end−to−end interactions
- epsilon_j_a = relative well depth of type J for side−to−side interactions
- epsilon_j_b = relative well depth of type J for face−to−face interactions
- epsilon_j_c = relative well depth of type J for end−to−end interactions
- cutoff (distance units)

The parameters used depend on the type of the interacting particles, i.e. ellipsoids or LJ spheres. The type of a particle is determined by the diameters specified for its 3 shape paramters. If all 3 shape parameters = 0.0, then the particle is treated as an LJ sphere. The epsilon_i_* or epsilon_j_* parameters are ignored for LJ spheres. If the 3 shape paraemters are > 0.0, then the particle is treated as an ellipsoid (even if the 3 parameters are equal to each other).

A12 specifies the energy prefactor which depends on the types of the two interacting particles.

For ellipsoid/ellipsoid interactions, the interaction is computed by the formulas in the supplementary docuement

referenced above. A12 is the Hamaker constant as described in (Everaers). In LJ units:

$$A_{12} = 4\pi^2 \epsilon_{\mathrm{LJ}} (\rho \sigma^3)^2$$

where rho gives the number density of the spherical particles composing the ellipsoids and epsilon_LJ determines the interaction strength of the spherical particles.

For ellipsoid/LJ sphere interactions, the interaction is also computed by the formulas in the supplementary docuement referenced above. A12 has a modifed form (see here for details):

$$A_{12} = 4\pi^2 \epsilon_{\mathrm{LJ}} (\rho \sigma^3)$$

For ellipsoid/LJ sphere interactions, a correction to the distance−of−closest approach equation has been implemented to reduce the error from two particles of disparate sizes; see this supplementary document.

For LJ sphere/LJ sphere interactions, the interaction is computed using the standard Lennard−Jones formula, which is much cheaper to compute than the ellipsoidal formulas. A12 is used as epsilon in the standard LJ formula:

$$A_{12} = \epsilon_{\mathrm{LJ}}$$

and the specified *sigma* is used as the sigma in the standard LJ formula.

When one of both of the interacting particles are ellipsoids, then *sigma* specifies the diameter of the continuous distribution of constituent particles within each ellipsoid used to model the RE−squared potential. Note that this is a different meaning for *sigma* than the pair_style gayberne potential uses.

The epsilon_i and epsilon_j coefficients are defined for atom types, not for pairs of atom types. Thus, in a series of pair_coeff commands, they only need to be specified once for each atom type.

Specifically, if any of epsilon_i_a, epsilon_i_b, epsilon_i_c are non−zero, the three values are assigned to atom type I. If all the epsilon_i values are zero, they are ignored. If any of epsilon_j_a, epsilon_j_b, epsilon_j_c are non−zero, the three values are assigned to atom type J. If all three epsilon_i values are zero, they are ignored. Thus the typical way to define the epsilon_i and epsilon_j coefficients is to list their values in "pair_coeff I J" commands when I = J, but set them to 0.0 when I != J. If you do list them when I != J, you should insure they are consistent with their values in other pair_coeff commands.

Note that if this potential is being used as a sub−style of pair_style hybrid, and there is no "pair_coeff I I" setting made for RE−squared for a particular type I (because I−I interactions are computed by another hybrid pair potential), then you still need to insure the epsilon a,b,c coefficients are assigned to that type in a "pair_coeff I J" command.

For large uniform molecules it has been shown that the epsilon_*_* energy parameters are approximately representable in terms of local contact curvatures (Everaers):

$$\epsilon_a = \sigma \cdot \frac{a}{b \cdot c}; \epsilon_b = \sigma \cdot \frac{b}{a \cdot c}; \epsilon_c = \sigma \cdot \frac{c}{a \cdot b}$$

where a, b, and c give the particle diameters.

The last coefficient is optional. If not specified, the global cutoff specified in the pair_style command is used.

Styles with a *cuda*, *gpu*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in this section of the manual. The accelerated styles take the same arguments and should produce the same results, except for round−off and precision issues.

These accelerated styles are part of the "user−cuda", "gpu", and "opt" packages respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the −suffix command−line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See this section of the manual for more instructions on how to use the accelerated styles effectively.

Mixing, shift, table, tail correction, restart, rRESPA info:

For atom type pairs I,J and I != J, the epsilon and sigma coefficients and cutoff distance can be mixed, but only for sphere pairs. The default mix value is *geometric*. See the "pair_modify" command for details. Other type pairs cannot be mixed, due to the different meanings of the energy prefactors used to calculate the interactions and the implicit dependence of the ellipsoid−sphere interaction on the equation for the Hamaker constant presented here. Mixing of sigma and epsilon followed by calculation of the energy prefactors using the equations above is recommended.

This pair styles supports the pair_modify shift option for the energy of the Lennard−Jones portion of the pair interaction, but only for sphere−sphere interactions. There is no shifting performed for ellipsoidal interactions due to the anisotropic dependence of the interaction.

The pair_modify table option is not relevant for this pair style.

This pair style does not support the pair_modify tail option for adding long−range tail corrections to energy and pressure.

This pair style writes its information to binary restart files, so pair_style and pair_coeff commands do not need to be specified in an input script that reads a restart file.

This pair style can only be used via the *pair* keyword of the run_style respa command. It does not support the *inner*, *middle*, *outer* keywords of the run_style command.

Restrictions:

This style is part of the "asphere" package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

This pair style requires that atoms be ellipsoids as defined by the atom_style ellipsoid command.

Particles acted on by the potential can be extended aspherical or spherical particles, or point particles. Spherical particles have all 3 of their shape parameters equal to each other. Point particles have all 3 of their shape parameters equal to 0.0.

The distance−of−closest−approach approximation used by LAMMPS becomes less accurate when high−aspect ratio ellipsoids are used.

Related commands:

pair_coeff, fix nve/asphere, compute temp/asphere, pair_style gayberne

Default: none

(Everaers) Everaers and Ejtehadi, Phys Rev E, 67, 041710 (2003).

(Berardi) Babadi, Ejtehadi, Everaers, J Comp Phys, 219, 770−779 (2006).

pair_style soft command

Syntax:

```
pair_style soft cutoff
```

- cutoff = global cutoff for soft interactions (distance units)

Examples:

```
pair_style soft 2.5
pair_coeff * * 10.0
pair_coeff 1 1 10.0 3.0

pair_style soft 2.5
pair_coeff * * 0.0
variable prefactor equal ramp(0,30)
fix 1 all adapt 1 pair soft a * * v_prefactor
```

Description:

Style *soft* computes pairwise interactions with the formula

$$E = A \left[1 + \cos\left(\frac{\pi r}{r_c}\right)\right] \qquad r < r_c$$

It is useful for pushing apart overlapping atoms, since it does not blow up as r goes to 0. A is a pre−factor that can be made to vary in time from the start to the end of the run (see discussion below), e.g. to start with a very soft potential and slowly harden the interactions over time. Rc is the cutoff. See the fix nve/limit command for another way to push apart overlapping atoms.

The following coefficients must be defined for each pair of atom types via the pair_coeff command as in the examples above, or in the data file or restart files read by the read_data or read_restart commands, or by mixing as described below:

- A (energy units)
- cutoff (distance units)

The last coefficient is optional. If not specified, the global soft cutoff is used.

IMPORTANT NOTE: The syntax for pair_coeff with a single A coeff is different in the current version of LAMMPS than in older versions which took two values, Astart and Astop, to ramp between them. This functionality is now available in a more general form through the fix adapt command, as explained below. Note that if you use an old input script and specify Astart and Astop without a cutoff, then LAMMPS will interpret that as A and a cutoff, which is probabably not what you want.

The fix adapt command can be used to vary A for one or more pair types over the course of a simulation, in which case pair_coeff settings for A must still be specified, but will be overridden. For example these commands will vary the prefactor A for all pairwise interactions from 0.0 at the beginning to 30.0 at the end of a run:

```
variable prefactor equal ramp(0,30)
fix 1 all adapt 1 pair soft a * * v_prefactor
```

Note that a formula defined by an equal−style variable can use the current timestep, elapsed time in the current run, elapsed time since the beginning of a series of runs, as well as access other variables.

Mixing, shift, table, tail correction, restart, rRESPA info:

For atom type pairs I,J and I != J, the A coefficient and cutoff distance for this pair style can be mixed. A is always mixed via a *geometric* rule. The cutoff is mixed according to the pair_modify mix value. The default mix value is *geometric*. See the "pair_modify" command for details.

This pair style does not support the pair_modify shift option, since the pair interaction goes to 0.0 at the cutoff.

The pair_modify table and tail options are not relevant for this pair style.

This pair style writes its information to binary restart files, so pair_style and pair_coeff commands do not need to be specified in an input script that reads a restart file.

This pair style can only be used via the *pair* keyword of the run_style respa command. It does not support the *inner*, *middle*, *outer* keywords.

Restrictions: none

Related commands:

pair_coeff, fix nve/limit, fix adapt

Default: none

pair_style command

Syntax:

```
pair_style style args
```

- style = one of the styles from the list below
- args = arguments used by a particular style

Examples:

```
pair_style lj/cut 2.5
pair_style eam/alloy
pair_style hybrid lj/charmm/coul/long 10.0 eam
pair_style table linear 1000
pair_style none
```

Description:

Set the formula(s) LAMMPS uses to compute pairwise interactions. In LAMMPS, pair potentials are defined between pairs of atoms that are within a cutoff distance and the set of active interactions typically changes over time. See the bond_style command to define potentials between pairs of bonded atoms, which typically remain in place for the duration of a simulation.

In LAMMPS, pairwise force fields encompass a variety of interactions, some of which include many−body effects, e.g. EAM, Stillinger−Weber, Tersoff, REBO potentials. They are still classified as "pairwise" potentials because the set of interacting atoms changes with time (unlike molecular bonds) and thus a neighbor list is used to find nearby interacting atoms.

Hybrid models where specified pairs of atom types interact via different pair potentials can be setup using the *hybrid* pair style.

The coefficients associated with a pair style are typically set for each pair of atom types, and are specified by the pair_coeff command or read from a file by the read_data or read_restart commands.

The pair_modify command sets options for mixing of type I−J interaction coefficients and adding energy offsets or tail corrections to Lennard−Jones potentials. Details on these options as they pertain to individual potentials are described on the doc page for the potential. Likewise, info on whether the potential information is stored in a restart file is listed on the potential doc page.

In the formulas listed for each pair style, E is the energy of a pairwise interaction between two atoms separated by a distance r. The force between the atoms is the negative derivative of this expression.

If the pair_style command has a cutoff argument, it sets global cutoffs for all pairs of atom types. The distance(s) can be smaller or larger than the dimensions of the simulation box.

Typically, the global cutoff value can be overridden for a specific pair of atom types by the pair_coeff command. The pair style settings (including global cutoffs) can be changed by a subsequent pair_style command using the same style. This will reset the cutoffs for all atom type pairs, including those previously set explicitly by a pair_coeff command. The exceptions to this are that pair_style *table* and *hybrid* settings cannot be reset. A new pair_style command for these styles will wipe out all previously specified pair_coeff values.

Here is an alphabetic list of pair styles defined in LAMMPS. Click on the style to display the formula it computes, arguments specified in the pair_style command, and coefficients specified by the associated pair_coeff command:

- pair_style none − turn off pairwise interactions
- pair_style hybrid − multiple styles of pairwise interactions
- pair_style hybrid/overlay − multiple styles of superposed pairwise interactions

- pair_style adp − angular dependent potential (ADP) of Mishin
- pair_style airebo − AIREBO potential of Stuart
- pair_style born − Born−Mayer−Huggins potential
- pair_style born/coul/long − Born−Mayer−Huggins with long−range Coulomb
- pair_style buck − Buckingham potential
- pair_style buck/coul/cut − Buckingham with cutoff Coulomb
- pair_style buck/coul/long − Buckingham with long−range Coulomb
- pair_style colloid − integrated colloidal potential
- pair_style coul/cut − cutoff Coulombic potential
- pair_style coul/debye − cutoff Coulombic potential with Debye screening
- pair_style coul/long − long−range Coulombic potential
- pair_style dipole/cut − point dipoles with cutoff
- pair_style dpd − dissipative particle dynamics (DPD)
- pair_style dpd/tstat − DPD thermostatting
- pair_style dsmc − Direct Simulation Monte Carlo (DSMC)
- pair_style eam − embedded atom method (EAM)
- pair_style eam/alloy − alloy EAM
- pair_style eam/fs − Finnis−Sinclair EAM
- pair_style eim − embedded ion method (EIM)
- pair_style gauss − Gaussian potential
- pair_style gayberne − Gay−Berne ellipsoidal potential
- pair_style gran/hertz/history − granular potential with Hertzian interactions
- pair_style gran/hooke − granular potential with history effects
- pair_style gran/hooke/history − granular potential without history effects
- pair_style hbond/dreiding/lj − DREIDING hydrogen bonding LJ potential
- pair_style hbond/dreiding/morse − DREIDING hydrogen bonding Morse potential
- pair_style lj/charmm/coul/charmm − CHARMM potential with cutoff Coulomb
- pair_style lj/charmm/coul/charmm/implicit − CHARMM for implicit solvent
- pair_style lj/charmm/coul/long − CHARMM with long−range Coulomb
- pair_style lj/class2 − COMPASS (class 2) force field with no Coulomb
- pair_style lj/class2/coul/cut − COMPASS with cutoff Coulomb
- pair_style lj/class2/coul/long − COMPASS with long−range Coulomb
- pair_style lj/cut − cutoff Lennard−Jones potential with no Coulomb
- pair_style lj/cut/coul/cut − LJ with cutoff Coulomb
- pair_style lj/cut/coul/debye − LJ with Debye screening added to Coulomb
- pair_style lj/cut/coul/long − LJ with long−range Coulomb
- pair_style lj/cut/coul/long/tip4p − LJ with long−range Coulomb for TIP4P water
- pair_style lj/expand − Lennard−Jones for variable size particles
- pair_style lj/gromacs − GROMACS−style Lennard−Jones potential
- pair_style lj/gromacs/coul/gromacs − GROMACS−style LJ and Coulombic potential
- pair_style lj/smooth − smoothed Lennard−Jones potential
- pair_style lj96/cut − Lennard−Jones 9/6 potential
- pair_style lubricate − hydrodynamic lubrication forces
- pair_style meam − modified embedded atom method (MEAM)

- pair_style morse − Morse potential
- pair_style peri/lps − peridynamic LPS potential
- pair_style peri/pmb − peridynamic PMB potential
- pair_style reax − ReaxFF potential
- pair_style rebo − 2nd generation REBO potential of Brenner
- pair_style resquared − Everaers RE−Squared ellipsoidal potential
- pair_style soft − Soft (cosine) potential
- pair_style sw − Stillinger−Weber 3−body potential
- pair_style table − tabulated pair potential
- pair_style tersoff − Tersoff 3−body potential
- pair_style tersoff/zbl − Tersoff/ZBL 3−body potential
- pair_style yukawa − Yukawa potential
- pair_style yukawa/colloid − screened Yukawa potential for finite−size particles

There are also additional pair styles submitted by users which are included in the LAMMPS distribution. The list of these with links to the individual styles are given in the pair section of this page.

There are also additional accelerated pair styles included in the LAMMPS distribution for faster performance on CPUs and GPUs. The list of these with links to the individual styles are given in the pair section of this page.

Restrictions:

This command must be used before any coefficients are set by the pair_coeff, read_data, or read_restart commands.

Some pair styles are part of specific packages. They are only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info on packages. The doc pages for individual pair potentials tell if it is part of a package.

Related commands:

pair_coeff, read_data, pair_modify, kspace_style, dielectric, pair_write

Default:

```
pair_style none
```

pair_style sw command

Syntax:

```
pair_style sw
```

Examples:

```
pair_style sw
pair_coeff * * si.sw Si
pair_coeff * * GaN.sw Ga N Ga
```

Description:

The *sw* style computes a 3−body Stillinger−Weber potential for the energy E of a system of atoms as

$$
\begin{aligned}
E &= \sum_i \sum_{j>i} \phi_2(r_{ij}) + \sum_i \sum_{j \neq i} \sum_{k>j} \phi_3(r_{ij}, r_{ik}, \theta_{ijk}) \\
\phi_2(r_{ij}) &= A_{ij}\epsilon_{ij} \left[B_{ij}(\frac{\sigma_{ij}}{r_{ij}})^{p_{ij}} - (\frac{\sigma_{ij}}{r_{ij}})^{q_{ij}} \right] \exp\left(\frac{\sigma_{ij}}{r_{ij} - a_{ij}\sigma_{ij}}\right) \\
\phi_3(r_{ij}, r_{ik}, \theta_{ijk}) &= \lambda_{ijk}\epsilon_{ijk} \left[\cos\theta_{ijk} - \cos\theta_{0ijk}\right]^2 \exp\left(\frac{\gamma_{ij}\sigma_{ij}}{r_{ij} - a_{ij}\sigma_{ij}}\right) \exp\left(\frac{\gamma_{ik}\sigma_{ik}}{r_{ik} - a_{ik}\sigma_{ik}}\right)
\end{aligned}
$$

where phi2 is a two−body term and phi3 is a three−body term. The summations in the formula are over all neighbors J and K of atom I within a cutoff distance = a*sigma.

Only a single pair_coeff command is used with the *sw* style which specifies a Stillinger−Weber potential file with parameters for all needed elements. These are mapped to LAMMPS atom types by specifying N additional arguments after the filename in the pair_coeff command, where N is the number of LAMMPS atom types:

- filename
- N element names = mapping of SW elements to atom types

As an example, imagine a file SiC.sw has Stillinger−Weber values for Si and C. If your LAMMPS simulation has 4 atoms types and you want the 1st 3 to be Si, and the 4th to be C, you would use the following pair_coeff command:

```
pair_coeff * * SiC.sw Si Si Si C
```

The 1st 2 arguments must be * * so as to span all LAMMPS atom types. The first three Si arguments map LAMMPS atom types 1,2,3 to the Si element in the SW file. The final C argument maps LAMMPS atom type 4 to the C element in the SW file. If a mapping value is specified as NULL, the mapping is not performed. This can be used when a *sw* potential is used as part of the *hybrid* pair style. The NULL values are placeholders for atom types that will be used with other potentials.

Stillinger−Weber files in the *potentials* directory of the LAMMPS distribution have a ".sw" suffix. Lines that are not blank or comments (starting with #) define parameters for a triplet of elements. The parameters in a single entry correspond to the two−body and three−body coefficients in the formula above:

- element 1 (the center atom in a 3−body interaction)
- element 2
- element 3
- epsilon (energy units)
- sigma (distance units)
- a
- lambda
- gamma
- costheta0
- A
- B
- p
- q
- tol

The A, B, p, and q parameters are used only for two−body interactions. The lambda and costheta0 parameters are used only for three−body interactions. The epsilon, sigma and a parameters are used for both two−body and three−body interactions. gamma is used only in the three−body interactions, but is defined for pairs of atoms. The non−annotated parameters are unitless.

LAMMPS introduces an additional performance−optimization parameter tol that is used for both two−body and three−body interactions. In the Stillinger−Weber potential, the interaction energies become negligibly small at atomic separations substantially less than the theoretical cutoff distances. LAMMPS therefore defines a virtual cutoff distance based on a user defined tolerance tol. The use of the virtual cutoff distance in constructing atom neighbor lists can significantly reduce the neighbor list sizes and therefore the computational cost. LAMMPS provides a *tol* value for each of the three−body entries so that they can be separately controlled. If tol = 0.0, then the standard Stillinger−Weber cutoff is used.

The Stillinger−Weber potential file must contain entries for all the elements listed in the pair_coeff command. It can also contain entries for additional elements not being used in a particular simulation; LAMMPS ignores those entries.

For a single−element simulation, only a single entry is required (e.g. SiSiSi). For a two−element simulation, the file must contain 8 entries (for SiSiSi, SiSiC, SiCSi, SiCC, CSiSi, CSiC, CCSi, CCC), that specify SW parameters for all permutations of the two elements interacting in three−body configurations. Thus for 3 elements, 27 entries would be required, etc.

As annotated above, the first element in the entry is the center atom in a three−body interaction. Thus an entry for SiCC means a Si atom with 2 C atoms as neighbors. The parameter values used for the two−body interaction come from the entry where the 2nd and 3rd elements are the same. Thus the two−body parameters for Si interacting with C, comes from the SiCC entry. The three−body parameters can in principle be specific to the three elements of the configuration. In the literature, however, the three−body parameters are usually defined by simple formulas involving two sets of pair−wise parameters, corresponding to the ij and ik pairs, where i is the center atom. The user must ensure that the correct combining rule is used to calculate the values of the threebody parameters for alloys. Note also that the function phi3 contains two exponential screening factors with parameter values from the ij pair and ik pairs. So phi3 for a C atom bonded to a Si atom and a second C atom will depend on the three−body parameters for the CSiC entry, and also on the two−body parameters for the CCC and CSiSi entries. Since the order of the two neighbors is arbitrary, the threebody parameters for entries CSiC and CCSi should be the same. Similarly, the two−body parameters for entries SiCC and CSiSi should also be the same. The parameters used only for two−body interactions (A, B, p, and q) in entries whose 2nd and 3rd element are different (e.g. SiCSi) are not used for anything and can be set to 0.0 if desired. This is also true for the parameters in phi3 that are taken from the ij and ik pairs (sigma, a, gamma)

Mixing, shift, table, tail correction, restart, rRESPA info:

For atom type pairs I,J and I != J, where types I and J correspond to two different element types, mixing is performed by LAMMPS as described above from values in the potential file.

This pair style does not support the pair_modify shift, table, and tail options.

This pair style does not write its information to binary restart files, since it is stored in potential files. Thus, you need to re−specify the pair_style and pair_coeff commands in an input script that reads a restart file.

This pair style can only be used via the *pair* keyword of the run_style respa command. It does not support the *inner*, *middle*, *outer* keywords.

Restrictions:

This pair style is part of the "manybody" package. It is only enabled if LAMMPS was built with that package (which it is by default). See the Making LAMMPS section for more info.

This pair style requires the newton setting to be "on" for pair interactions.

The Stillinger−Weber potential files provided with LAMMPS (see the potentials directory) are parameterized for metal units. You can use the SW potential with any LAMMPS units, but you would need to create your own SW potential file with coefficients listed in the appropriate units if your simulation doesn't use "metal" units.

Related commands:

pair_coeff

Default: none

(Stillinger) Stillinger and Weber, Phys Rev B, 31, 5262 (1985).

pair_style table command

Syntax:

```
pair_style table style N
```

- style = *lookup* or *linear* or *spline* or *bitmap* = method of interpolation
- N = use N values in *lookup*, *linear*, *spline* tables
- N = use 2^N values in *bitmap* tables

Examples:

```
pair_style table linear 1000
pair_style table bitmap 12
pair_coeff * 3 morse.table ENTRY1
pair_coeff * 3 morse.table ENTRY1 7.0
```

Description:

Style *table* creates interpolation tables of length *N* from pair potential and force values listed in a file(s) as a function of distance. The files are read by the pair_coeff command.

The interpolation tables are created by fitting cubic splines to the file values and interpolating energy and force values at each of *N* distances. During a simulation, these tables are used to interpolate energy and force values as needed. The interpolation is done in one of 4 styles: *lookup*, *linear*, *spline*, or *bitmap*.

For the *lookup* style, the distance between 2 atoms is used to find the nearest table entry, which is the energy or force.

For the *linear* style, the pair distance is used to find 2 surrounding table values from which an energy or force is computed by linear interpolation.

For the *spline* style, a cubic spline coefficients are computed and stored at each of the *N* values in the table. The pair distance is used to find the appropriate set of coefficients which are used to evaluate a cubic polynomial which computes the energy or force.

For the *bitmap* style, the N means to create interpolation tables that are 2^N in length. (Wolff) and a linear interpolation is performed between adjacent table values.

The following coefficients must be defined for each pair of atoms types via the pair_coeff command as in the examples above.

- filename
- keyword
- cutoff (distance units)

The filename specifies a file containing tabulated energy and force values. The keyword specifies a section of the file. The cutoff is an optional coefficient. If not specified, the outer cutoff in the table itself (see below) will be used to build an interpolation table that extend to the largest tabulated distance. If specified, only file values up to the cutoff are used to create the interpolation table. The format of this file is described below.

Here are some guidelines for using the pair_style table command to best effect:

- Vary the number of table points; you may need to use more than you think to get good resolution.
- Always use the pair_write command to produce a plot of what the final interpolated potential looks like. This can show up interpolation "features" you may not like.
- Start with the linear style; it's the style least likely to have problems.
- Use *N* in the pair_style command equal to the "N" in the tabulation file, and use the "RSQ" or "BITMAP" parameter, so additional interpolation is not needed. See discussion below.
- Use as large an inner cutoff as possible. This avoids fitting splines to very steep parts of the potential.

The format of a tabulated file is as follows (without the parenthesized comments):

```
# Morse potential for Fe   (one or more comment or blank lines)

MORSE_FE                   (keyword is first text on line)
N 500 R 1.0 10.0           (N, R, RSQ, BITMAP, FPRIME parameters)
                           (blank)
1 1.0 25.5 102.34          (index, r, energy, force)
2 1.02 23.4 98.5
...
500 10.0 0.001 0.003
```

A section begins with a non−blank line whose 1st character is not a "#"; blank lines or lines starting with "#" can be used as comments between sections. The first line begins with a keyword which identifies the section. The line can contain additional text, but the initial text must match the argument specified in the pair_coeff command. The next line lists (in any order) one or more parameters for the table. Each parameter is a keyword followed by one or more numeric values.

The parameter "N" is required and its value is the number of table entries that follow. Note that this may be different than the *N* specified in the pair_style table command. Let Ntable = *N* in the pair_style command, and Nfile = "N" in the tabulated file. What LAMMPS does is a preliminary interpolation by creating splines using the Nfile tabulated values as nodal points. It uses these to interpolate as needed to generate energy and force values at Ntable different points. The resulting tables of length Ntable are then used as described above, when computing energy and force for individual pair distances. This means that if you want the interpolation tables of length Ntable to match exactly what is in the tabulated file (with effectively no preliminary interpolation), you should set Ntable = Nfile, and use the "RSQ" or "BITMAP" parameter.

All other parameters are optional. If "R" or "RSQ" or "BITMAP" does not appear, then the distances in each line of the table are used as−is to perform spline interpolation. In this case, the table values can be spaced in *r* uniformly or however you wish to position table values in regions of large gradients.

If used, the parameters "R" or "RSQ" are followed by 2 values *rlo* and *rhi*. If specified, the distance associated with each energy and force value is computed from these 2 values (at high accuracy), rather than using the (low−accuracy) value listed in each line of the table. For "R", distances uniformly spaced between *rlo* and *rhi* are computed; for "RSQ", squared distances uniformly spaced between *rlo*rlo* and *rhi*rhi* are computed.

If used, the parameter "BITMAP" is also followed by 2 values *rlo* and *rhi*. These values, along with the "N" value determine the ordering of the N lines that follow and what distance is associated with each. This ordering is complex, so it is not documented here, since this file is typically produced by the pair_write command with its *bitmap* option. When the table is in BITMAP format, the "N" parameter in the file must be equal to 2^M where M is the value specified in the pair_style command. Also, a cutoff parameter cannot be used as an optional 3rd argument in the pair_coeff command; the entire table extent as specified in the file must be used.

If used, the parameter "FPRIME" is followed by 2 values *fplo* and *fphi* which are the derivative of the force at the innermost and outermost distances listed in the table. These values are needed by the spline construction routines. If not specified by the "FPRIME" parameter, they are estimated (less accurately) by the first 2 and last 2 force values in the table. This parameter is not used by BITMAP tables.

Following a blank line, the next N lines list the tabulated values. On each line, the 1st value is the index from 1 to N, the 2nd value is r (in distance units), the 3rd value is the energy (in energy units), and the 4th is the force (in force units). The r values must increase from one line to the next (unless the BITMAP parameter is specified).

Note that one file can contain many sections, each with a tabulated potential. LAMMPS reads the file section by section until it finds one that matches the specified keyword.

Mixing, shift, table, tail correction, restart, rRESPA info:

This pair style does not support mixing. Thus, coefficients for all I,J pairs must be specified explicitly.

The pair_modify shift, table, and tail options are not relevant for this pair style.

This pair style writes the settings for the "pair_style table" command to binary restart files, so a pair_style command does not need to specified in an input script that reads a restart file. However, the coefficient information is not stored in the restart file, since it is tabulated in the potential files. Thus, pair_coeff commands do need to be specified in the restart input script.

This pair style can only be used via the *pair* keyword of the run_style respa command. It does not support the *inner*, *middle*, *outer* keywords.

Restrictions: none

Related commands:

pair_coeff

Default: none

(Wolff) Wolff and Rudd, Comp Phys Comm, 120, 200−32 (1999).

pair_style tersoff command

Syntax:

```
pair_style tersoff
```

Examples:

```
pair_style tersoff
pair_coeff * * Si.tersoff Si
pair_coeff * * SiC.tersoff Si C Si
```

Description:

The *tersoff* style computes a 3−body Tersoff potential (Tersoff_1) for the energy E of a system of atoms as

$$
\begin{aligned}
E &= \frac{1}{2}\sum_i \sum_{j \neq i} V_{ij} \\
V_{ij} &= f_C(r_{ij}) \left[f_R(r_{ij}) + b_{ij} f_A(r_{ij}) \right] \\
f_C(r) &= \begin{cases} 1 & : \quad r < R - D \\ \frac{1}{2} - \frac{1}{2} \sin\left(\frac{\pi}{2} \frac{r-R}{D} \right) & : \quad R - D < r < R + D \\ 0 & : \quad r > R + D \end{cases} \\
f_R(r) &= A \exp(-\lambda_1 r) \\
f_A(r) &= -B \exp(-\lambda_2 r) \\
b_{ij} &= \left(1 + \beta^n {\zeta_{ij}}^n\right)^{-\frac{1}{2n}} \\
\zeta_{ij} &= \sum_{k \neq i,j} f_C(r_{ik}) g(\theta_{ijk}) \exp\left[{\lambda_3}^m (r_{ij} - r_{ik})^m \right] \\
g(\theta) &= \gamma_{ijk} \left(1 + \frac{c^2}{d^2} - \frac{c^2}{\left[d^2 + (\cos\theta - \cos\theta_0)^2 \right]} \right)
\end{aligned}
$$

where f_R is a two−body term and f_A includes three−body interactions. The summations in the formula are over all neighbors J and K of atom I within a cutoff distance = R + D.

Only a single pair_coeff command is used with the *tersoff* style which specifies a Tersoff potential file with parameters for all needed elements. These are mapped to LAMMPS atom types by specifying N additional arguments after the filename in the pair_coeff command, where N is the number of LAMMPS atom types:

- filename
- N element names = mapping of Tersoff elements to atom types

As an example, imagine the SiC.tersoff file has Tersoff values for Si and C. If your LAMMPS simulation has 4 atoms types and you want the 1st 3 to be Si, and the 4th to be C, you would use the following pair_coeff command:

```
pair_coeff * * SiC.tersoff Si Si Si C
```

The 1st 2 arguments must be * * so as to span all LAMMPS atom types. The first three Si arguments map LAMMPS atom types 1,2,3 to the Si element in the Tersoff file. The final C argument maps LAMMPS atom type 4 to the C element in the Tersoff file. If a mapping value is specified as NULL, the mapping is not performed. This can be used when a *tersoff* potential is used as part of the *hybrid* pair style. The NULL values are placeholders for atom types that will be used with other potentials.

Tersoff files in the *potentials* directory of the LAMMPS distribution have a ".tersoff" suffix. Lines that are not blank or comments (starting with #) define parameters for a triplet of elements. The parameters in a single entry correspond to coefficients in the formula above:

- element 1 (the center atom in a 3−body interaction)
- element 2 (the atom bonded to the center atom)
- element 3 (the atom influencing the 1−2 bond in a bond−order sense)
- m
- gamma
- lambda3 (1/distance units)
- c
- d
- costheta0 (can be a value < −1 or > 1)
- n
- beta
- lambda2 (1/distance units)
- B (energy units)
- R (distance units)
- D (distance units)
- lambda1 (1/distance units)
- A (energy units)

The n, beta, lambda2, B, lambda1, and A parameters are only used for two−body interactions. The m, gamma, lambda3, c, d, and costheta0 parameters are only used for three−body interactions. The R and D parameters are used for both two−body and three−body interactions. The non−annotated parameters are unitless. The value of m must be 3 or 1.

The Tersoff potential file must contain entries for all the elements listed in the pair_coeff command. It can also contain entries for additional elements not being used in a particular simulation; LAMMPS ignores those entries.

For a single−element simulation, only a single entry is required (e.g. SiSiSi). For a two−element simulation, the file must contain 8 entries (for SiSiSi, SiSiC, SiCSi, SiCC, CSiSi, CSiC, CCSi, CCC), that specify Tersoff parameters for all permutations of the two elements interacting in three−body configurations. Thus for 3 elements, 27 entries would be required, etc.

As annotated above, the first element in the entry is the center atom in a three−body interaction and it is bonded to the 2nd atom and the bond is influenced by the 3rd atom. Thus an entry for SiCC means Si bonded to a C with another C atom influencing the bond. Thus three−body parameters for SiCSi and SiSiC entries will not, in general, be the same. The parameters used for the two−body interaction come from the entry where the 2nd element is repeated. Thus the two−body parameters for Si interacting with C, comes from the SiCC entry.

The parameters used for a particular three−body interaction come from the entry with the corresponding three elements. The parameters used only for two−body interactions (n, beta, lambda2, B, lambda1, and A) in entries whose 2nd and 3rd element are different (e.g. SiCSi) are not used for anything and can be set to 0.0 if desired.

Note that the twobody parameters in entries such as SiCC and CSiSi are often the same, due to the common use of

symmetric mixing rules, but this is not always the case. For example, the beta and n parameters in Tersoff_2 (Tersoff_2) are not symmetric.

We chose the above form so as to enable users to define all commonly used variants of the Tersoff potential. In particular, our form reduces to the original Tersoff form when m = 3 and gamma = 1, while it reduces to the form of Albe et al. when beta = 1 and m = 1. Note that in the current Tersoff implementation in LAMMPS, m must be specified as either 3 or 1. Tersoff used a slightly different but equivalent form for alloys, which we will refer to as Tersoff_2 potential (Tersoff_2).

LAMMPS parameter values for Tersoff_2 can be obtained as follows: gamma = 1, just as for Tersoff_1, but now lambda3 = 0 and the value of m has no effect. The parameters for species i and j can be calculated using the Tersoff_2 mixing rules:

$$
\begin{aligned}
\lambda_1^{i,j} &= \frac{1}{2}(\lambda_1^i + \lambda_1^j) \\
\lambda_2^{i,j} &= \frac{1}{2}(\lambda_2^i + \lambda_2^j) \\
A_{i,j} &= (A_i A_j)^{1/2} \\
B_{i,j} &= \chi_{ij}(B_i B_j)^{1/2} \\
R_{i,j} &= (R_i R_j)^{1/2} \\
S_{i,j} &= (S_i S_j)^{1/2}
\end{aligned}
$$

Tersoff_2 parameters R and S must be converted to the LAMMPS parameters R and D (R is different in both forms), using the following relations: R=(R'+S')/2 and D=(S'−R')/2, where the primes indicate the Tersoff_2 parameters.

In the potentials directory, the file SiCGe.tersoff provides the LAMMPS parameters for Tersoff's various versions of Si, as well as his alloy parameters for Si, C, and Ge. This file can be used for pure Si, (three different versions), pure C, pure Ge, binary SiC, and binary SiGe. LAMMPS will generate an error if this file is used with any combination involving C and Ge, since there are no entries for the GeC interactions (Tersoff did not publish parameters for this cross−interaction.) Tersoff files are also provided for the SiC alloy (SiC.tersoff) and the GaN (GaN.tersoff) alloys.

Many thanks to Rutuparna Narulkar, David Farrell, and Xiaowang Zhou for helping clarify how Tersoff parameters for alloys have been defined in various papers.

Mixing, shift, table, tail correction, restart, rRESPA info:

For atom type pairs I,J and I != J, where types I and J correspond to two different element types, mixing is performed by LAMMPS as described above from values in the potential file.

This pair style does not support the pair_modify shift, table, and tail options.

This pair style does not write its information to binary restart files, since it is stored in potential files. Thus, you need to re−specify the pair_style and pair_coeff commands in an input script that reads a restart file.

This pair style can only be used via the *pair* keyword of the run_style respa command. It does not support the *inner*, *middle*, *outer* keywords.

Restrictions:

This pair style is part of the "manybody" package. It is only enabled if LAMMPS was built with that package (which it is by default). See the Making LAMMPS section for more info.

This pair style requires the newton setting to be "on" for pair interactions.

The Tersoff potential files provided with LAMMPS (see the potentials directory) are parameterized for metal units. You can use the Tersoff potential with any LAMMPS units, but you would need to create your own Tersoff potential file with coefficients listed in the appropriate units if your simulation doesn't use "metal" units.

Related commands:

pair_coeff

Default: none

(Tersoff_1) J. Tersoff, Phys Rev B, 37, 6991 (1988).

(Albe) J. Nord, K. Albe, P. Erhart, and K. Nordlund, J. Phys.: Condens. Matter, 15, 5649(2003).

(Tersoff_2) J. Tersoff, Phys Rev B, 39, 5566 (1989); errata (PRB 41, 3248)

pair_style tersoff/zbl command

Syntax:

```
pair_style tersoff/zbl
```

Examples:

```
pair_style tersoff/zbl
pair_coeff * * SiC.tersoff.zbl Si C Si
```

Description:

The *tersoff/zbl* style computes a 3−body Tersoff potential (Tersoff_1) with a close−separation pairwise modification based on a Coulomb potential and the Ziegler−Biersack−Littmark universal screening function (ZBL), giving the energy E of a system of atoms as

$$
\begin{aligned}
E &= \frac{1}{2}\sum_i \sum_{j \neq i} V_{ij} \\
V_{ij} &= (1 - f_F(r_{ij}))V_{ij}^{ZBL} + f_F(r_{ij})V_{ij}^{Tersoff} \\
f_F(r_{ij}) &= \frac{1}{1 + e^{-A_F(r_{ij} - r_C)}}
\end{aligned}
$$

$$
\begin{aligned}
V_{ij}^{ZBL} &= \frac{1}{4\pi\epsilon_0}\frac{Z_1 Z_2\, e^2}{r_{ij}}\phi(r_{ij}/a) \\
a &= \frac{0.8854\, a_0}{Z_1^{0.23} + Z_2^{0.23}} \\
\phi(x) &= 0.1818e^{-3.2x} + 0.5099e^{-0.9423x} + 0.2802e^{-0.4029x} + 0.02817e^{-0.2016x}
\end{aligned}
$$

$$
\begin{aligned}
V_{ij}^{Tersoff} &= f_C(r_{ij})\left[f_R(r_{ij}) + b_{ij} f_A(r_{ij})\right] \\
f_C(r) &= \begin{cases} 1 & : \quad r < R - D \\ \frac{1}{2} - \frac{1}{2}\sin\left(\frac{\pi}{2}\frac{r-R}{D}\right) & : \quad R - D < r < R + D \\ 0 & : \quad r > R + D \end{cases} \\
f_R(r) &= A\exp(-\lambda_1 r) \\
f_A(r) &= -B\exp(-\lambda_2 r) \\
b_{ij} &= (1 + \beta^n {\zeta_{ij}}^n)^{-\frac{1}{2n}} \\
\zeta_{ij} &= \sum_{k \neq i,j} f_C(r_{ik}) g(\theta_{ijk}) \exp\left[{\lambda_3}^3 (r_{ij} - r_{ik})^m\right] \\
g(\theta) &= \gamma_{ijk}\left(1 + \frac{c^2}{d^2} - \frac{c^2}{[d^2 + (\cos\theta - \cos\theta_0)^2]}\right)
\end{aligned}
$$

The f_F term is a fermi−like function used to smoothly connect the ZBL repulsive potential with the Tersoff potential. There are 2 parameters used to adjust it: A_F and r_C. A_F controls how "sharp" the transition is between the two, and r_C is essentially the cutoff for the ZBL potential.

For the ZBL portion, there are two terms. The first is the Coulomb repulsive term, with Z1, Z2 as the number of protons in each nucleus, e as the electron charge (1 for metal and real units) and epsilon0 as the permittivity of vacuum. The second part is the ZBL universal screening function, with a0 being the Bohr radius (typically 0.529 Angstroms), and the remainder of the coefficients provided by the original paper. This screening function should be applicable to most systems. However, it is only accurate for small separations (i.e. less than 1 Angstrom).

For the Tersoff portion, f_R is a two−body term and f_A includes three−body interactions. The summations in the formula are over all neighbors J and K of atom I within a cutoff distance = R + D.

Only a single pair_coeff command is used with the *tersoff/zbl* style which specifies a Tersoff/ZBL potential file with parameters for all needed elements. These are mapped to LAMMPS atom types by specifying N additional arguments after the filename in the pair_coeff command, where N is the number of LAMMPS atom types:

- filename
- N element names = mapping of Tersoff/ZBL elements to atom types

As an example, imagine the SiC.tersoff.zbl file has Tersoff/ZBL values for Si and C. If your LAMMPS simulation has 4 atoms types and you want the 1st 3 to be Si, and the 4th to be C, you would use the following pair_coeff command:

```
pair_coeff * * SiC.tersoff Si Si Si C
```

The 1st 2 arguments must be * * so as to span all LAMMPS atom types. The first three Si arguments map LAMMPS atom types 1,2,3 to the Si element in the Tersoff/ZBL file. The final C argument maps LAMMPS atom type 4 to the C element in the Tersoff/ZBL file. If a mapping value is specified as NULL, the mapping is not performed. This can be used when a *tersoff/zbl* potential is used as part of the *hybrid* pair style. The NULL values are placeholders for atom types that will be used with other potentials.

Tersoff/ZBL files in the *potentials* directory of the LAMMPS distribution have a ".tersoff.zbl" suffix. Lines that are not blank or comments (starting with #) define parameters for a triplet of elements. The parameters in a single entry correspond to coefficients in the formula above:

- element 1 (the center atom in a 3−body interaction)
- element 2 (the atom bonded to the center atom)
- element 3 (the atom influencing the 1−2 bond in a bond−order sense)
- m
- gamma
- lambda3 (1/distance units)
- c
- d
- costheta0 (can be a value < −1 or > 1)
- n
- beta
- lambda2 (1/distance units)
- B (energy units)
- R (distance units)
- D (distance units)
- lambda1 (1/distance units)
- A (energy units)

- Z_i
- Z_j
- ZBLcut (distance units)
- ZBLexpscale (1/distance units)

The n, beta, lambda2, B, lambda1, and A parameters are only used for two−body interactions. The m, gamma, lambda3, c, d, and costheta0 parameters are only used for three−body interactions. The R and D parameters are used for both two−body and three−body interactions. The Z_i,Z_j, ZBLcut, ZBLexpscale parameters are used in the ZBL repulsive portion of the potential and in the Fermi−like function. The non−annotated parameters are unitless. The value of m must be 3 or 1.

The Tersoff/ZBL potential file must contain entries for all the elements listed in the pair_coeff command. It can also contain entries for additional elements not being used in a particular simulation; LAMMPS ignores those entries.

For a single−element simulation, only a single entry is required (e.g. SiSiSi). For a two−element simulation, the file must contain 8 entries (for SiSiSi, SiSiC, SiCSi, SiCC, CSiSi, CSiC, CCSi, CCC), that specify Tersoff parameters for all permutations of the two elements interacting in three−body configurations. Thus for 3 elements, 27 entries would be required, etc.

As annotated above, the first element in the entry is the center atom in a three−body interaction and it is bonded to the 2nd atom and the bond is influenced by the 3rd atom. Thus an entry for SiCC means Si bonded to a C with another C atom influencing the bond. Thus three−body parameters for SiCSi and SiSiC entries will not, in general, be the same. The parameters used for the two−body interaction come from the entry where the 2nd element is repeated. Thus the two−body parameters for Si interacting with C, comes from the SiCC entry. By symmetry, the twobody parameters in the SiCC and CSiSi entries should thus be the same. The parameters used for a particular three−body interaction come from the entry with the corresponding three elements. The parameters used only for two−body interactions (n, beta, lambda2, B, lambda1, and A) in entries whose 2nd and 3rd element are different (e.g. SiCSi) are not used for anything and can be set to 0.0 if desired.

We chose the above form so as to enable users to define all commonly used variants of the Tersoff portion of the potential. In particular, our form reduces to the original Tersoff form when m = 3 and gamma = 1, while it reduces to the form of Albe et al. when beta = 1 and m = 1. Note that in the current Tersoff implementation in LAMMPS, m must be specified as either 3 or 1. Tersoff used a slightly different but equivalent form for alloys, which we will refer to as Tersoff_2 potential (Tersoff_2).

LAMMPS parameter values for Tersoff_2 can be obtained as follows: gamma = 1, just as for Tersoff_1, but now lambda3 = 0 and the value of m has no effect. The parameters for species i and j can be calculated using the Tersoff_2 mixing rules:

$$
\begin{aligned}
\lambda_1^{i,j} &= \frac{1}{2}(\lambda_1^i + \lambda_1^j) \\
\lambda_2^{i,j} &= \frac{1}{2}(\lambda_2^i + \lambda_2^j) \\
A_{i,j} &= (A_i A_j)^{1/2} \\
B_{i,j} &= \chi_{ij}(B_i B_j)^{1/2} \\
R_{i,j} &= (R_i R_j)^{1/2} \\
S_{i,j} &= (S_i S_j)^{1/2}
\end{aligned}
$$

Values not shown are determined by the first atom type. Finally, the Tersoff_2 parameters R and S must be converted to the LAMMPS parameters R and D (R is different in both forms), using the following relations: R=(R'+S')/2 and D=(S'−R')/2, where the primes indicate the Tersoff_2 parameters.

In the potentials directory, the file SiCGe.tersoff provides the LAMMPS parameters for Tersoff's various versions of Si, as well as his alloy parameters for Si, C, and Ge. This file can be used for pure Si, (three different versions), pure C, pure Ge, binary SiC, and binary SiGe. LAMMPS will generate an error if this file is used with any combination involving C and Ge, since there are no entries for the GeC interactions (Tersoff did not publish parameters for this cross−interaction.) Tersoff files are also provided for the SiC alloy (SiC.tersoff) and the GaN (GaN.tersoff) alloys.

Many thanks to Rutuparna Narulkar, David Farrell, and Xiaowang Zhou for helping clarify how Tersoff parameters for alloys have been defined in various papers. Also thanks to Ram Devanathan for providing the base ZBL implementation.

Mixing, shift, table, tail correction, restart, rRESPA info:

For atom type pairs I,J and I != J, where types I and J correspond to two different element types, mixing is performed by LAMMPS as described above from values in the potential file.

This pair style does not support the pair_modify shift, table, and tail options.

This pair style does not write its information to binary restart files, since it is stored in potential files. Thus, you need to re−specify the pair_style and pair_coeff commands in an input script that reads a restart file.

This pair style can only be used via the *pair* keyword of the run_style respa command. It does not support the *inner*, *middle*, *outer* keywords.

Restrictions:

This pair style is part of the "manybody" package. It is only enabled if LAMMPS was built with that package (which it is by default). See the Making LAMMPS section for more info.

This pair style requires the newton setting to be "on" for pair interactions.

The Tersoff/ZBL potential files provided with LAMMPS (see the potentials directory) are parameterized for metal units. You can use the Tersoff potential with any LAMMPS units, but you would need to create your own Tersoff potential file with coefficients listed in the appropriate units if your simulation doesn't use "metal" units.

Related commands:

pair_coeff

Default: none

(Tersoff_1) J. Tersoff, Phys Rev B, 37, 6991 (1988).

(ZBL) J.F. Ziegler, J.P. Biersack, U. Littmark, 'Stopping and Ranges of Ions in Matter' Vol 1, 1985, Pergamon Press.

(Albe) J. Nord, K. Albe, P. Erhartand K. Nordlund, J. Phys.: Condens. Matter, 15, 5649(2003).

(Tersoff_2) J. Tersoff, Phys Rev B, 39, 5566 (1989)

pair_write command

Syntax:

```
pair_write itype jtype N style inner outer file keyword Qi Qj
```

- itype,jtype = 2 atom types
- N = # of values
- style = *r* or *rsq* or *bitmap*
- inner,outer = inner and outer cutoff (distance units)
- file = name of file to write values to
- keyword = section name in file for this set of tabulated values
- Qi,Qj = 2 atom charges (charge units) (optional)

Examples:

```
pair_write 1 3 500 r 1.0 10.0 table.txt LJ
pair_write 1 1 1000 rsq 2.0 8.0 table.txt Yukawa_1_1 −0.5 0.5
```

Description:

Write energy and force values to a file as a function of distance for the currently defined pair potential. This is useful for plotting the potential function or otherwise debugging its values. If the file already exists, the table of values is appended to the end of the file to allow multiple tables of energy and force to be included in one file.

The energy and force values are computed at distances from inner to outer for 2 interacting atoms of type itype and jtype, using the appropriate pair_coeff coefficients. If the style is *r*, then N distances are used, evenly spaced in r; if the style is *rsq*, N distances are used, evenly spaced in r^2.

For example, for N = 7, style = *r*, inner = 1.0, and outer = 4.0, values are computed at r = 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0.

If the style is *bitmap*, then 2^N values are written to the file in a format and order consistent with how they are read in by the pair_coeff command for pair style *table*. For reasonable accuracy in a bitmapped table, choose N >= 12, an *inner* value that is smaller than the distance of closest approach of 2 atoms, and an *outer* value <= cutoff of the potential.

If the pair potential is computed between charged atoms, the charges of the pair of interacting atoms can optionally be specified. If not specified, values of Qi = Qj = 1.0 are used.

The file is written in the format used as input for the pair_style *table* option with *keyword* as the section name. Each line written to the file lists an index number (1−N), a distance (in distance units), an energy (in energy units), and a force (in force units).

Restrictions:

All force field coefficients for pair and other kinds of interactions must be set before this command can be invoked.

Due to how the pairwise force is computed, an inner value > 0.0 must be specified even if the potential has a finite

value at r = 0.0.

For EAM potentials, the pair_write command only tabulates the pairwise portion of the potential, not the embedding portion.

Related commands:

pair_style, pair_coeff

Default: none

pair_style yukawa command

Syntax:

```
pair_style yukawa kappa cutoff
```

- kappa = screening length (inverse distance units)
- cutoff = global cutoff for Yukawa interactions (distance units)

Examples:

```
pair_style yukawa 2.0 2.5
pair_coeff 1 1 100.0 2.3
pair_coeff * * 100.0
```

Description:

Style *yukawa* computes pairwise interactions with the formula

$$E = A\frac{e^{-\kappa r}}{r} \qquad r < r_c$$

Rc is the cutoff.

The following coefficients must be defined for each pair of atoms types via the pair_coeff command as in the examples above, or in the data file or restart files read by the read_data or read_restart commands, or by mixing as described below:

- A (energy*distance units)
- cutoff (distance units)

The last coefficient is optional. If not specified, the global yukawa cutoff is used.

Mixing, shift, table, tail correction, restart, rRESPA info:

For atom type pairs I,J and I != J, the A coefficient and cutoff distance for this pair style can be mixed. A is an energy value mixed like a LJ epsilon. The default mix value is *geometric*. See the "pair_modify" command for details.

This pair style supports the pair_modify shift option for the energy of the pair interaction.

The pair_modify table option is not relevant for this pair style.

This pair style does not support the pair_modify tail option for adding long−range tail corrections to energy and pressure.

This pair style writes its information to binary restart files, so pair_style and pair_coeff commands do not need to be specified in an input script that reads a restart file.

This pair style can only be used via the *pair* keyword of the run_style respa command. It does not support the *inner*, *middle*, *outer* keywords.

Restrictions: none

Related commands:

pair_coeff

Default: none

pair_style yukawa/colloid command

Syntax:

```
pair_style yukawa/colloid kappa cutoff
```

- kappa = screening length (inverse distance units)
- cutoff = global cutoff for colloidal Yukawa interactions (distance units)

Examples:

```
pair_style yukawa/colloid 2.0 2.5
pair_coeff 1 1 100.0 2.3
pair_coeff * * 100.0
```

Description:

Style *yukawa/colloid* computes pairwise interactions with the formula

$$E = \frac{A}{\kappa} e^{-\kappa(r-(r_i+r_j))} \qquad r < r_c$$

where Ri and Rj are the radii of the two particles and Rc is the cutoff.

In contrast to pair_style yukawa, this functional form arises from the Coulombic interaction between two colloid particles, screened due to the presence of an electrolyte. Pair_style yukawa is a screened Coulombic potential between two point−charges and uses no such approximation.

This potential applies to nearby particle pairs for which the Derjagin approximation holds, meaning h << Ri + Rj, where h is the surface−to−surface separation of the two particles.

When used in combination with pair_style colloid, the two terms become the so−called DLVO potential, which combines electrostatic repulsion and van der Waals attraction.

The following coefficients must be defined for each pair of atoms types via the pair_coeff command as in the examples above, or in the data file or restart files read by the read_data or read_restart commands, or by mixing as described below:

- A (energy/distance units)
- cutoff (distance units)

The prefactor A is determined from the relationship between surface charge and surface potential due to the presence of electrolyte. Note that the A for this potential style has different units than the A used in pair_style yukawa. For low surface potentials, i.e. less than about 25 mV, A can be written as:

```
A = 2 * PI * R*eps*eps0 * kappa * psi^2
```

where

- R = colloid radius (distance units)
- eps0 = permittivity of free space (charge^2/energy/distance units)
- eps = relative permittivity of fluid medium (dimensionless)
- kappa = inverse screening length (1/distance units)
- psi = surface potential (energy/charge units)

The last coefficient is optional. If not specified, the global yukawa/colloid cutoff is used.

Mixing, shift, table, tail correction, restart, rRESPA info:

For atom type pairs I,J and I != J, the A coefficient and cutoff distance for this pair style can be mixed. A is an energy value mixed like a LJ epsilon. The default mix value is *geometric*. See the "pair_modify" command for details.

This pair style supports the pair_modify shift option for the energy of the pair interaction.

The pair_modify table option is not relevant for this pair style.

This pair style does not support the pair_modify tail option for adding long−range tail corrections to energy and pressure.

This pair style writes its information to binary restart files, so pair_style and pair_coeff commands do not need to be specified in an input script that reads a restart file.

This pair style can only be used via the *pair* keyword of the run_style respa command. It does not support the *inner*, *middle*, *outer* keywords.

Restrictions:

This style is part of the "colloid" package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

This pair style requires that atoms be finite−size spheres with a diameter, as defined by the atom_style sphere command.

Per−particle polydispersity is not yet supported by this pair style; per−type polydispersity is allowed. This means all particles of the same type must have the same diameter. Each type can have a different diameter.

Related commands:

pair_coeff

Default: none

prd command

Syntax:

```
prd N t_event n_dephase t_dephase t_correlate compute−ID seed keyword value ...
```

- N = # of timesteps to run (not including dephasing/quenching)
- t_event = timestep interval between event checks
- n_dephase = number of velocity randomizations to perform in each dephase run
- t_dephase = number of timesteps to run dynamics after each velocity randomization during dephase
- t_correlate = number of timesteps within which 2 consecutive events are considered to be correlated
- compute−ID = ID of the compute used for event detection
- random_seed = random # seed (positive integer)
- zero or more keyword/value pairs may be appended
- keyword = *min* or *temp* or *vel*

```
  min values = etol ftol maxiter maxeval
      etol = stopping tolerance for energy, used in quenching
      ftol = stopping tolerance for force, used in quenching
      maxiter = max iterations of minimize, used in quenching
      maxeval = max number of force/energy evaluations, used in quenching
    temp value = Tdephase
      Tdephase = target temperature for velocity randomization, used in dephasing
    vel values = loop dist
      loop = all or local or geom, used in dephasing
      dist = uniform or gaussian, used in dephasing
```

Examples:

```
prd 5000 100 10 10 100 1 54982
prd 5000 100 10 10 100 1 54982 min 0.1 0.1 100 200
```

Description:

Run a parallel replica dynamics (PRD) simulation using multiple replicas of a system. One or more replicas can be used.

PRD is described in this paper by Art Voter. It is a method for performing accelerated dynamics that is suitable for infrequent−event systems that obey first−order kinetics. A good overview of accelerated dynamics methods for such systems in given in this review paper from the same group. To quote from the paper: "The dynamical evolution is characterized by vibrational excursions within a potential basin, punctuated by occasional transitions between basins." The transition probability is characterized by p(t) = k*exp(−kt) where k is the rate constant. Running multiple replicas gives an effective enhancement in the timescale spanned by the multiple simulations, while waiting for an event to occur.

Each replica runs on a partition of one or more processors. Processor partitions are defined at run−time using the −partition command−line switch; see this section of the manual. Note that if you have MPI installed, you can run a multi−replica simulation with more replicas (partitions) than you have physical processors, e.g you can run a 10−replica simulation on one or two processors. For PRD, this makes little sense, since this offers no effective parallel speed−up in searching for infrequent events. See this section of the manual for further discussion.

When a PRD simulation is performed, it is assumed that each replica is running the same model, though

LAMMPS does not check for this. I.e. the simulation domain, the number of atoms, the interaction potentials, etc should be the same for every replica.

A PRD run has several stages, which are repeated each time an "event" occurs in one of the replicas, as defined below. The logic for a PRD run is as follows:

```
while (time remains):
  dephase for n_dephase*t_dephase steps
  until (event occurs on some replica):
    run dynamics for t_event steps
    quench
    check for uncorrelated event on any replica
  until (no correlated event occurs):
    run dynamics for t_correlate steps
    quench
    check for correlated event on this replica
  event replica shares state with all replicas
```

Before this loop begins, the state of the system on replica 0 is shared with all replicas, so that all replicas begin from the same initial state. The first potential energy basin is identified by quenching (an energy minimization, see below) the initial state and storing the resulting coordinates for reference.

In the first stage, dephasing is performed by each replica independently to eliminate correlations between replicas. This is done by choosing a random set of velocities, based on the *random_seed* that is specified, and running *t_dephase* timesteps of dynamics. This is repeated *n_dephase* times. If the *temp* keyword is not specified, the target temperature for velocity randomization for each replica is the current temperature of that replica. Otherwise, it is the specified *Tdephase* temperature. The style of velocity randomization is controlled using the keyword *vel* with arguments that have the same meaning as their counterparts in the velocity command.

In the second stage, each replica runs dynamics continuously, stopping every *t_event* steps to check if a transition event has occurred. This check is performed by quenching the system and comparing the resulting atom coordinates to the coordinates from the previous basin. The first time through the PRD loop, the "previous basin" is the set of quenched coordinates from the initial state of the system.

A quench is an energy minimization and is performed by whichever algorithm has been defined by the min_style command. Minimization parameters may be set via the min_modify command and by the *min* keyword of the PRD command. The latter are the settings that would be used with the minimize command. Note that typically, you do not need to perform a highly−converged minimization to detect a transition event.

The event check is performed by a compute with the specified *compute−ID*. Currently there is only one compute that works with the PRD commmand, which is the compute event/displace command. Other event−checking computes may be added. Compute event/displace checks whether any atom in the compute group has moved further than a specified threshold distance. If so, an "event" has occurred.

In the third stage, the replica on which the event occurred (event replica) continues to run dynamics to search for correlated events. This is done by running dynamics for *t_correlate* steps, quenching every *t_event* steps, and checking if another event has occurred. The first time no correlated event occurs, the final state of the event replica is shared with all replicas, the new basin reference coordinates are updated with the quenched state, and the outer loop begins again. While the replica event is searching for correlated events, all the other replicas also run dynamics and event checking with the same schedule, but the final states are always overwritten by the state of the event replica.

Four kinds of output can be generated during a PRD run: event statistics, thermodynamic output by each replica, dump files, and restart files.

When running with multiple partitions (each of which is a replica in this case), the print−out to the screen and master log.lammps file is limited to event statistics. Note that if a PRD run is performed on only a single replica then the event statistics will be intermixed with the usual thermodynamic output discussed below.

The quantities printed each time an event occurs are the timestep, CPU time, clock, event number, a correlation flag, the number of coincident events, and the replica number of the chosen event.

The timestep is the usual LAMMPS timestep, except that time does not advance during dephasing or quenches, but only during dynamics. Note that are two kinds of dynamics in the PRD loop listed above. The first is when all replicas are performing independent dynamics. The second is when correlated events are being searched for and only one replica is running dynamics.

The CPU time is the total processor time since the start of the PRD run.

The clock is the same as the timestep except that it advances by M steps every timestep during the first kind of dynamics when the M replicas are running independently. The clock represents the real time that effectively elapses during a PRD simulation of *N* steps on M replicas. If most of the PRD run is spent in the second stage of the loop above, searching for infrequent events, then the clock will advance nearly N*M steps. Note the clock time between events will be drawn from p(t).

The event number is a counter that increments with each event, whether it is uncorrelated or correlated.

The correlation flag will be 0 when an uncorrelated event occurs during the second stage of the loop listed above, i.e. when all replicas are running independently. The correlation flag will be 1 when a correlated event occurs during the third stage of the loop listed above, i.e. when only one replica is running dynamics.

When more than one replica detects an event at the end of the second stage, then one of them is chosen at random. The number of coincident events is the number of replicas that detected an event. Normally, we expect this value to be 1. If it is often greater than 1, then either the number of replicas is too large, or *t_event* is too large.

The replica number is the ID of the replica (from 0 to M−1) that found the event.

When running on multiple partitions, LAMMPS produces additional log files for each partition, e.g. log.lammps.0, log.lammps.1, etc. For the PRD command, these contain the thermodynamic output for each replica. You will see short runs and minimizations corresponding to the dynamics and quench operations of the loop listed above. The timestep will be reset aprpopriately depending on whether the operation advances time or not.

After the PRD command completes, timing statistics for the PRD run are printed in each replica's log file, giving a breakdown of how much CPU time was spent in each stage (dephasing, dynamics, quenching, etc).

Any dump files defined in the input script, will be written to during a PRD run at timesteps corresponding to both uncorrelated and correlated events. This means the the requested dump frequency in the dump command is ignored. There will be one dump file (per dump command) created for all partitions.

The atom coordinates of the dump snapshot are those of the minimum energy configuration resulting from quenching following a transition event. The timesteps written into the dump files correspond to the timestep at which the event occurred and NOT the clock. A dump snapshot corresponding to the initial minimum state used for event detection is written to the dump file at the beginning of each PRD run.

If the restart command is used, a single restart file for all the partitions is generated, which allows a PRD run to be continued by a new input script in the usual manner.

The restart file is generated at the end of the loop listed above. If no correlated events are found, this means it contains a snapshot of the system at time T + *t_correlate*, where T is the time at which the uncorrelated event occurred. If correlated events were found, then it contains a snapshot of the system at time T + *t_correlate*, where T is the time of the last correlated event.

The restart frequency specified in the restart command is interpreted differently when performing a PRD run. It does not mean the timestep interval between restart files. Instead it means an event interval for uncorrelated events. Thus a frequency of 1 means write a restart file every time an uncorrelated event occurs. A frequency of 10 means write a restart file every 10th uncorrelated event.

When an input script reads a restart file from a previous PRD run, the new script can be run on a different number of replicas or processors. However, it is assumed that *t_correlate* in the new PRD command is the same as it was previously. If not, the calculation of the "clock" value for the first event in the new run will be slightly off.

Restrictions:

This command can only be used if LAMMPS was built with the "replica" package. See the Making LAMMPS section for more info on packages.

N and *t_correlate* settings must be integer multiples of *t_event*.

Runs restarted from restart file written during a PRD run will not produce identical results due to changes in the random numbers used for dephasing.

This command cannot be used when any fixes are defined that keep track of elapsed time to perform time−dependent operations. Examples include the "ave" fixes such as fix ave/spatial. Also fix dt/reset and fix deposit.

Related commands:

compute event/displace, min_modify, min_style, run_style, minimize, velocity, temper, neb, tad

Default:

The option defaults are *min* = 0.1 0.1 40 50, no *temp* setting, and *vel* = *geom gaussian*.

(Voter) Voter, Phys Rev B, 57, 13985 (1998).

(Voter2) Voter, Montalenti, Germann, Annual Review of Materials Research 32, 321 (2002).

print command

Syntax:

```
print str
```

- str = text string to print, which may contain variables

Examples:

```
print "Done with equilibration"
print Vol=$v
print "The system volume is now $v"
print 'The system volume is now $v'
```

Description:

Print a text string to the screen and logfile. One line of output is generated. If the string has white space in it (spaces, tabs, etc), then you must enclose it in quotes so that it is treated as a single argument. If variables are included in the string, they will be evaluated and their current values printed.

If you want the print command to be executed multiple times (with changing variable values), there are 3 options. First, consider using the fix print command, which will print a string periodically during a simulation. Second, the print command can be used as an argument to the *every* option of the run command. Third, the print command could appear in a section of the input script that is looped over (see the jump and next commands).

See the variable command for a description of *equal* style variables which are typically the most useful ones to use with the print command. Equal−style variables can calculate formulas involving mathematical operations, atom properties, group properties, thermodynamic properties, global values calculated by a compute or fix, or references to other variables.

Restrictions: none

Related commands:

fix print, variable

Default: none

processors command

Syntax:

```
processors Px Py Pz
```

- Px,Py,Pz = # of processors in each dimension of a 3d grid

Examples:

```
processors 2 4 4
processors * * 5
processors * 1 10
```

Description:

Specify how processors are mapped as a 3d logical grid to the global simulation box, namely Px by Py by Pz.

Any of the Px, Py, Pz parameters can be specified with an asterisk "*", which means LAMMPS will choose the number of processors in that dimension. It will do this based on the size and shape of the global simulation box so as to minimize the surface−to−volume ratio of each processor's sub−domain.

Since LAMMPS does not load−balance by changing the grid of 3d processors on−the−fly, this command can be used to override the LAMMPS default if it is known to be sub−optimal for a particular problem. For example, a problem where the atom's extent will change dramatically in a particular dimension over the course of the simulation.

The product of Px, Py, Pz must equal P, the total # of processors LAMMPS is running on. For a 2d simulation, Pz must equal 1. If multiple partitions are being used then P is the number of processors in this partition; see this section for an explanation of the −partition command−line switch.

Note that if you run on a large, prime number of processors P, then a grid such as 1 x P x 1 will be required, which may incur extra communication costs.

Restrictions:

This command cannot be used after the simulation box is defined by a read_data or create_box command. It can be used before a restart file is read to change the 3d processor grid from what is specified in the restart file.

Related commands: none

Default:

Px Py Pz = * * *

read_data command

Syntax:

```
read_data file
```

- file = name of data file to read in

Examples:

```
read_data data.lj
read_data ../run7/data.polymer.gz
```

Description:

Read in a data file containing information LAMMPS needs to run a simulation. The file can be ASCII text or a gzipped text file (detected by a .gz suffix). This is one of 3 ways to specify initial atom coordinates; see the read_restart and create_atoms commands for alternative methods.

The structure of the data file is important, though many settings and sections are optional or can come in any order. See the examples directory for sample data files for different problems.

A data file has a header and a body. The header appears first. The first line of the header is always skipped; it typically contains a description of the file. Then lines are read one at a time. Lines can have a trailing comment starting with '#' that is ignored. If the line is blank (only whitespace after comment is deleted), it is skipped. If the line contains a header keyword, the corresponding value(s) is read from the line. If it doesn't contain a header keyword, the line begins the body of the file.

The body of the file contains zero or more sections. The first line of a section has only a keyword. The next line is skipped. The remaining lines of the section contain values. The number of lines depends on the section keyword as described below. Zero or more blank lines can be used between sections. Sections can appear in any order, with a few exceptions as noted below.

The formatting of individual lines in the data file (indentation, spacing between words and numbers) is not important except that header and section keywords (e.g. atoms, xlo xhi, Masses, Bond Coeffs) must be capitalized as shown and can't have extra white space between their words − e.g. two spaces or a tab between "Bond" and "Coeffs" is not valid.

These are the recognized header keywords. Header lines can come in any order. The value(s) are read from the beginning of the line. Thus the keyword *atoms* should be in a line like "1000 atoms"; the keyword *ylo yhi* should be in a line like "−10.0 10.0 ylo yhi"; the keyword *xy xz yz* should be in a line like "0.0 5.0 6.0 xy xz yz". All these settings have a default value of 0, except the lo/hi box size defaults are −0.5 and 0.5. A line need only appear if the value is different than the default.

- *atoms* = # of atoms in system
- *bonds* = # of bonds in system
- *angles* = # of angles in system
- *dihedrals* = # of dihedrals in system
- *impropers* = # of impropers in system
- *atom types* = # of atom types in system

- *bond types* = # of bond types in system
- *angle types* = # of angle types in system
- *dihedral types* = # of dihedral types in system
- *improper types* = # of improper types in system
- *extra bond per atom* = leave space for this many new bonds per atom
- *ellipsoids* = # of ellipsoids in system
- *xlo xhi* = simulation box boundaries in x dimension
- *ylo yhi* = simulation box boundaries in y dimension
- *zlo zhi* = simulation box boundaries in z dimension
- *xy xz yz* = simulation box tilt factors for triclinic system

The initial simulation box size is determined by the lo/hi settings. In any dimension, the system may be periodic or non−periodic; see the boundary command.

If the *xy xz yz* line does not appear, LAMMPS will set up an axis−aligned (orthogonal) simulation box. If the line does appear, LAMMPS creates a non−orthogonal simulation domain shaped as a parallelepiped with triclinic symmetry. The parallelepiped has its "origin" at (xlo,ylo,zlo) and is defined by 3 edge vectors starting from the origin given by A = (xhi−xlo,0,0); B = (xy,yhi−ylo,0); C = (xz,yz,zhi−zlo). *Xy,xz,yz* can be 0.0 or positive or negative values and are called "tilt factors" because they are the amount of displacement applied to faces of an originally orthogonal box to transform it into the parallelepiped.

The tilt factors (xy,xz,yz) can not skew the box more than half the distance of the corresponding parallel box length. For example, if xlo = 2 and xhi = 12, then the x box length is 10 and the xy tilt factor must be between −5 and 5. Similarly, both xz and yz must be between −(xhi−xlo)/2 and +(yhi−ylo)/2. Note that this is not a limitation, since if the maximum tilt factor is 5 (as in this example), then configurations with tilt = ..., −15, −5, 5, 15, 25, ... are all geometrically equivalent.

See this section of the doc pages for a geometric description of triclinic boxes, as defined by LAMMPS, and how to transform these parameters to and from other commonly used triclinic representations.

When a triclinic system is used, the simulation domain must be periodic in any dimensions with a non−zero tilt factor, as defined by the boundary command. I.e. if the xy tilt factor is non−zero, then both the x and y dimensions must be periodic. Similarly, x and z must be periodic if xz is non−zero and y and z must be periodic if yz is non−zero. Also note that if your simulation will tilt the box, e.g. via the fix deform command, the simulation box must be defined as triclinic, even if the tilt factors are initially 0.0.

For 2d simulations, the *zlo zhi* values should be set to bound the z coords for atoms that appear in the file; the default of −0.5 0.5 is valid if all z coords are 0.0. For 2d triclinic simulations, the xz and yz tilt factors must be 0.0.

If the system is periodic (in a dimension), then atom coordinates can be outside the bounds (in that dimension); they will be remapped (in a periodic sense) back inside the box.

IMPORTANT NOTE: If the system is non−periodic (in a dimension), then all atoms in the data file must have coordinates (in that dimension) that are "greater than or equal to" the lo value and "less than or equal to" the hi value. If the non−periodic dimension is of style "fixed" (see the boundary command), then the atom coords must be strictly "less than" the hi value, due to the way LAMMPS assign atoms to processors. Note that you should not make the lo/hi values radically smaller/larger than the extent of the atoms. For example, if your atoms extend from 0 to 50, you should not specify the box bounds as −10000 and 10000. This is because LAMMPS uses the specified box size to layout the 3d grid of processors. A huge (mostly empty) box will be sub−optimal for performance when using "fixed" boundary conditions (see the boundary command). When using "shrink−wrap" boundary conditions (see the boundary command), a huge (mostly empty) box may cause a parallel simulation to

lose atoms the first time that LAMMPS shrink−wraps the box around the atoms.

The "extra bond per atom" setting should be used if new bonds will be added to the system when a simulation runs, e.g. by using the fix bond/create command. This will pre−allocate space in LAMMPS data structures for storing the new bonds.

The "ellipsoids setting is only used with atom_style ellipsoid and specifies how many of the atoms are finite−size ellipsoids; the remainder are point particles. See the discussion of ellipseflag and the *Ellipsoids* section below.

These are the section keywords for the body of the file.

- *Atoms, Velocities, Ellipsoids, Masses* = atom−property sections
- *Bonds, Angles, Dihedrals, Impropers* = molecular topology sections
- *Pair Coeffs, Bond Coeffs, Angle Coeffs, Dihedral Coeffs, Improper Coeffs* = force field sections
- *BondBond Coeffs, BondAngle Coeffs, MiddleBondTorsion Coeffs, EndBondTorsion Coeffs, AngleTorsion Coeffs, AngleAngleTorsion Coeffs, BondBond13 Coeffs, AngleAngle Coeffs* = class 2 force field sections

Each section is listed below in alphabetic order. The format of each section is described including the number of lines it must contain and rules (if any) for where it can appear in the data file.

Any individual line in the various sections can have a trailing comment starting with "#" for annotation purposes. E.g. in the Atoms section:

```
10 1 17 −1.0 10.0 5.0 6.0   # salt ion
```

Angle Coeffs section:

- one line per angle type
- line syntax: ID coeffs

  ```
  ID = angle type (1−N)
  coeffs = list of coeffs
  ```
- example:

  ```
  6 70 108.5 0 0
  ```

The number and meaning of the coefficients are specific to the defined angle style. See the angle_style and angle_coeff commands for details. Coefficients can also be set via the angle_coeff command in the input script.

AngleAngle Coeffs section:

- one line per improper type
- line syntax: ID coeffs

  ```
  ID = improper type (1−N)
  coeffs = list of coeffs (see improper_coeff)
  ```

AngleAngleTorsion Coeffs section:

- one line per dihedral type
- line syntax: ID coeffs

  ```
  ID = dihedral type (1−N)
  ```

```
coeffs = list of coeffs (see dihedral_coeff)
```

Angles section:

- one line per angle
- line syntax: ID type atom1 atom2 atom3

```
ID = number of angle (1-Nangles)
type = angle type (1-Nangletype)
atom1,atom2,atom3 = IDs of 1st,2nd,3rd atoms in angle
```

example:

```
2 2 17 29 430
```

The 3 atoms are ordered linearly within the angle. Thus the central atom (around which the angle is computed) is the atom2 in the list. E.g. H,O,H for a water molecule. The *Angles* section must appear after the *Atoms* section. All values in this section must be integers (1, not 1.0).

AngleTorsion Coeffs section:

- one line per dihedral type
- line syntax: ID coeffs

```
ID = dihedral type (1-N)
coeffs = list of coeffs (see dihedral_coeff)
```

Atoms section:

- one line per atom
- line syntax: depends on atom style

An *Atoms* section must appear in the data file if natoms > 0 in the header section. The atoms can be listed in any order. These are the line formats for each atom style in LAMMPS. As discussed below, each line can optionally have 3 flags (nx,ny,nz) appended to it, which indicate which image of a periodic simulation box the atom is in. These may be important to include for some kinds of analysis.

angle	atom-ID molecule-ID atom-type x y z
atomic	atom-ID atom-type x y z
bond	atom-ID molecule-ID atom-type x y z
charge	atom-ID atom-type q x y z
dipole	atom-ID atom-type q x y z mux muy muz
electron	atom-ID atom-type q spin eradius x y z
ellipsoid	atom-ID atom-type ellipsoidflag density x y z
full	atom-ID molecule-ID atom-type q x y z
molecular	atom-ID molecule-ID atom-type x y z
peri	atom-ID atom-type volume density x y z
sphere	atom-ID atom-type diameter density x y z
wavepacket	atom-ID atom-type charge spin eradius etag cs_re cs_im x y z
hybrid	atom-ID atom-type x y z sub-style1 sub-style2 ...

The keywords have these meanings:

- atom−ID = integer ID of atom
- molecule−ID = integer ID of molecule the atom belongs to
- atom−type = type of atom (1−Ntype)
- q = charge on atom (charge units)
- diameter = diameter of spherical atom (distance units)
- ellipsoidflag = 1 for ellipsoidal particles, 0 for point particles
- density = density of atom (mass/distance^3 units)
- volume = volume of atom (distance^3 units)
- x,y,z = coordinates of atom
- mux,muy,muz = components of dipole moment of atom (dipole units)
- spin = electron spin (+1/−1), 0 = nuclei, 2 = fixed−core, 3 = pseudo−cores (i.e. ECP)
- eradius = electron radius (or fixed−core radius)
- etag = integer ID of electron that each wavepacket belongs to
- cs_re,cs_im = real/imaginary parts of wavepacket coefficients

The units for these quantities depend on the unit style; see the units command for details.

For 2d simulations specify z as 0.0, or a value within the *zlo zhi* setting in the data file header.

The atom−ID is used to identify the atom throughout the simulation and in dump files. Normally, it is a unique value from 1 to Natoms for each atom. Unique values larger than Natoms can be used, but they will cause extra memory to be allocated on each processor, if an atom map array is used (see the atom_modify command). If an atom map array is not used (e.g. an atomic system with no bonds), and velocities are not assigned in the data file, and you don't care if unique atom IDs appear in dump files, then the atom−IDs can all be set to 0.

The molecule ID is a 2nd identifier attached to an atom. Normally, it is a number from 1 to N, identifying which molecule the atom belongs to. It can be 0 if it is an unbonded atom or if you don't care to keep track of molecule assignments.

The diameter specifies the size of a finite−size spherical particle. It can be set to 0.0, which means that atom is a point particle.

The ellipseflag determines whether the particle is a finite−size ellipsoid of finite size, or a point particle. Additional attributes must be defined for each ellipsoid in the *Ellipsoids* section.

Some pair styles and fixes and computes that operate on finite−size particles allow for a mixture of finite−size and point particles. See the doc pages of individual commands for details.

The density is used in conjunction with the particle volume for finite−size particles to set the mass of the particle as mass = density * volume. If the volume is 0.0, meaning a point particle, then the density value is used as the mass.

For atom_style hybrid, following the 5 initial values (ID,type,x,y,z), specific values for each sub−style must be listed. The order of the sub−styles is the same as they were listed in the atom_style command. The sub−style specific values are those that are not the 5 standard ones (ID,type,x,y,z). For example, for the "charge" sub−style, a "q" value would appear. For the "full" sub−style, a "molecule−ID" and "q" would appear. These are listed in the same order they appear as listed above. Thus if

```
atom_style hybrid charge sphere
```

were used in the input script, each atom line would have these fields:

```
atom-ID atom-type x y z q diameter density
```

Atom lines (all lines or none of them) can optionally list 3 trailing integer values: nx,ny,nz. For periodic dimensions, they specify which image of the simulation box the atom is considered to be in. An image of 0 means it is inside the box as defined. A value of 2 means add 2 box lengths to get the true value. A value of −1 means subtract 1 box length to get the true value. LAMMPS updates these flags as atoms cross periodic boundaries during the simulation. The flags can be output with atom snapshots via the dump command.

If nx,ny,nz values are not set in the data file, LAMMPS initializes them to 0. If image information is needed for later analysis and they are not all initially 0, it's important to set them correctly in the data file. Also, if you plan to use the replicate command to generate a larger system, these flags must be listed correctly for bonded atoms when the bond crosses a periodic boundary. I.e. the values of the image flags should be different by 1 (in the appropriate dimension) for the two atoms in such a bond.

Atom velocities and other atom quantities not defined above are set to 0.0 when the *Atoms* section is read. Velocities can be set later by a *Velocities* section in the data file or by a velocity or set command in the input script.

Bond Coeffs section:

- one line per bond type
- line syntax: ID coeffs

```
ID = bond type (1-N)
coeffs = list of coeffs
```

- example:

```
4 250 1.49
```

The number and meaning of the coefficients are specific to the defined bond style. See the bond_style and bond_coeff commands for details. Coefficients can also be set via the bond_coeff command in the input script.

BondAngle Coeffs section:

- one line per angle type
- line syntax: ID coeffs

```
ID = angle type (1-N)
coeffs = list of coeffs (see class 2 section of angle_coeff)
```

BondBond Coeffs section:

- one line per angle type
- line syntax: ID coeffs

```
ID = angle type (1-N)
coeffs = list of coeffs (see class 2 section of angle_coeff)
```

BondBond13 Coeffs section:

- one line per dihedral type
- line syntax: ID coeffs

```
ID = dihedral type (1-N)
```

```
   coeffs = list of coeffs (see class 2 section of dihedral_coeff)
```

Bonds section:

- one line per bond
- line syntax: ID type atom1 atom2

```
   ID = bond number (1-Nbonds)
   type = bond type (1-Nbondtype)
   atom1,atom2 = IDs of 1st,2nd atoms in bond
```

- example:

```
   12 3 17 29
```

The *Bonds* section must appear after the *Atoms* section. All values in this section must be integers (1, not 1.0).

Dihedral Coeffs section:

- one line per dihedral type
- line syntax: ID coeffs

```
   ID = dihedral type (1-N)
   coeffs = list of coeffs
```

- example:

```
   3 0.6 1 0 1
```

The number and meaning of the coefficients are specific to the defined dihedral style. See the dihedral_style and dihedral_coeff commands for details. Coefficients can also be set via the dihedral_coeff command in the input script.

Dihedrals section:

- one line per dihedral
- line syntax: ID type atom1 atom2 atom3 atom4

```
   ID = number of dihedral (1-Ndihedrals)
   type = dihedral type (1-Ndihedraltype)
   atom1,atom2,atom3,atom4 = IDs of 1st,2nd,3rd,4th atoms in dihedral
```

- example:

```
   12 4 17 29 30 21
```

The 4 atoms are ordered linearly within the dihedral. The *Dihedrals* section must appear after the *Atoms* section. All values in this section must be integers (1, not 1.0).

Ellipsoids section:

- one line per ellipsoid
- line syntax: atom-ID shapex shapey shapez quatw quati quatj quatk

```
   atom-ID = ID of atom which is an ellipsoid
   shapex,shapey,shapez = 3 diameters of ellipsoid (distance units)
   quatw,quati,quatj,quatk = quaternion components for orientation of atom
type = bond type (1-Nbondtype)
```

```
atom1,atom2 = IDs of 1st,2nd atoms in bond
```

- example:

```
12 3 17 29
```

The *Ellipsoids* section must appear if atom_style ellipsoid is used and any atoms are listed in the *Atoms* section with an ellipsoidflag = 1. The number of ellipsoids should be specified in the header section via the "ellipsoids" keyword.

The 3 shape values specify the 3 diameters or aspect ratios of a finite−size ellipsoidal particle, when it is oriented along the 3 coordinate axes. They must all be non−zero values.

The values *quatw*, *quati*, *quatj*, and *quatk* set the orientation of the atom as a quaternion (4−vector). Note that the shape attributes specify the aspect ratios of an ellipsoidal particle, which is oriented by default with its x−axis along the simulation box's x−axis, and similarly for y and z. If this body is rotated (via the right−hand rule) by an angle theta around a unit vector (a,b,c), then the quaternion that represents its new orientation is given by (cos(theta/2), a*sin(theta/2), b*sin(theta/2), c*sin(theta/2)). These 4 components are quatw, quati, quatj, and quatk as specified above. LAMMPS normalizes each atom's quaternion in case (a,b,c) is not specified as a unit vector.

The *Ellipsoids* section must appear after the *Atoms* section.

EndBondTorsion Coeffs section:

- one line per dihedral type
- line syntax: ID coeffs

```
ID = dihedral type (1−N)
coeffs = list of coeffs (see class 2 section of dihedral_coeff)
```

Improper Coeffs section:

- one line per improper type
- line syntax: ID coeffs

```
ID = improper type (1−N)
coeffs = list of coeffs
```

- example:

```
2 20 0.0548311
```

The number and meaning of the coefficients are specific to the defined improper style. See the improper_style and improper_coeff commands for details. Coefficients can also be set via the improper_coeff command in the input script.

Impropers section:

- one line per improper
- line syntax: ID type atom1 atom2 atom3 atom4

```
ID = number of improper (1−Nimpropers)
type = improper type (1−Nimpropertype)
atom1,atom2,atom3,atom4 = IDs of 1st,2nd,3rd,4th atoms in improper
```

- example:

```
12 3 17 29 13 100
```

The ordering of the 4 atoms determines the definition of the improper angle used in the formula for each improper style. See the doc pages for individual styles for details.

The *Impropers* section must appear after the *Atoms* section. All values in this section must be integers (1, not 1.0).

Masses section:

- one line per atom type
- line syntax: ID mass

```
ID = atom type (1-N)
mass = mass value
```

- example:

```
3 1.01
```

This defines the mass of each atom type. This can also be set via the mass command in the input script. This section cannot be used for atom styles that define a mass for individual atoms − e.g. atom_style sphere.

MiddleBondTorsion Coeffs section:

- one line per dihedral type
- line syntax: ID coeffs

```
ID = dihedral type (1-N)
coeffs = list of coeffs (see class 2 section of dihedral_coeff)
```

Pair Coeffs section:

- one line per atom type
- line syntax: ID coeffs

```
ID = atom type (1-N)
coeffs = list of coeffs
```

- example:

```
3 0.022 2.35197 0.022 2.35197
```

The number and meaning of the coefficients are specific to the defined pair style. See the pair_style and pair_coeff commands for details. Coefficients can also be set via the pair_coeff command in the input script.

Velocities section:

- one line per atom
- line syntax: depends on atom style

all styles except those listed	atom−ID vx vy vz
dipole	atom−ID vx vy vz wx wy wz
electron	atom−ID vx vy vz evel
ellipsoid	atom−ID vx vy vz lx ly lz

sphere	atom−ID vx vy vz wx wy wz

where the keywords have these meanings:

vx,vy,vz = translational velocity of atom lx,ly,lz = angular momentum of aspherical atom wx,wy,wz = angular velocity of spherical atom evel = electron radial velocity (0 for fixed−core):ul

The velocity lines can appear in any order. This section can only be used after an *Atoms* section. This is because the *Atoms* section must have assigned a unique atom ID to each atom so that velocities can be assigned to them.

Vx, vy, vz, and evel are in units of velocity. Lx, ly, lz are in units of angular momentum (distance−velocity−mass). Wx, Wy, Wz are in units of angular velocity (radians/time).

Translational velocities can also be set by the velocity command in the input script.

Restrictions:

To read gzipped data files, you must compile LAMMPS with the −DLAMMPS_GZIP option − see the Making LAMMPS section of the documentation.

Related commands:

read_restart, create_atoms

Default: none

read_restart command

Syntax:

```
read_restart file
```

- file = name of binary restart file to read in

Examples:

```
read_restart save.10000
read_restart restart.*
read_restart poly.*.%
```

Description:

Read in a previously saved simulation from a restart file. This allows continuation of a previous run. Information about what is stored in a restart file is given below.

Restart files are saved in binary format to enable exact restarts, meaning that the trajectories of a restarted run will precisely match those produced by the original run had it continued on. Several things can prevent exact restarts due to round−off effects, in which case the trajectories in the 2 runs will slowly diverge. These include running on a different number of processors or changing certain settings such as those set by the newton or processors commands. LAMMPS will issue a warning in these cases. Certain fixes will also not restart exactly, though they should provide statistically similar results. These include fix shake and fix langevin. If a restarted run is immediately different than the run which produced the restart file, it could be a LAMMPS bug, so consider reporting it if you think the behavior is wrong.

Because restart files are binary, they may not be portable to other machines. They can be converted to ASCII data files using the restart2data tool in the tools sub−directory of the LAMMPS distribution.

Similar to how restart files are written (see the write_restart and restart commands), the restart filename can contain two wild−card characters. If a "*" appears in the filename, the directory is searched for all filenames that match the pattern where "*" is replaced with a timestep value. The file with the largest timestep value is read in. Thus, this effectively means, read the latest restart file. It's useful if you want your script to continue a run from where it left off. See the run command and its "upto" option for how to specify the run command so it doesn't need to be changed either.

If a "%" character appears in the restart filename, LAMMPS expects a set of multiple files to exist. The restart and write_restart commands explain how such sets are created. Read_restart will first read a filename where "%" is replaced by "base". This file tells LAMMPS how many processors created the set. Read_restart then reads the additional files. For example, if the restart file was specified as save.% when it was written, then read_restart reads the files save.base, save.0, save.1, ... save.P−1, where P is the number of processors that created the restart file. The processors in the current LAMMPS simulation share the work of reading these files; each reads a roughly equal subset of the files. The number of processors which created the set can be different the number of processors in the current LAMMPS simulation. This can be a fast mode of input on parallel machines that support parallel I/O.

A restart file stores the following information about a simulation: units and atom style, simulation box size and shape and boundary settings, group definitions, per−type atom settings such as mass, per−atom attributes

including their group assignments and molecular topology attributes, force field styles and coefficients, and special_bonds settings. This means that commands for these quantities do not need to be re−specified in the input script that reads the restart file, though you can redefine settings after the restart file is read.

One exception is that some pair styles do not store their info in restart files. The doc pages for individual pair styles note if this is the case. This is also true of bond_style hybrid (and angle_style, dihedral_style, improper_style hybrid).

Information about kspace_style settings are not stored in the restart file. Hence if you wish to use an Ewald or PPPM solver, these commands must be re−issued after the restart file is read.

The list of fixes used for a simulation is not stored in the restart file. This means the new input script should specify all fixes it will use. Note that some fixes store an internal "state" which is written to the restart file. This allows the fix to continue on with its calculations in a restarted simulation. To re−enable such a fix, the fix command in the new input script must use the same fix−ID and group−ID as was used in the input script that wrote the restart file. If a match is found, LAMMPS prints a message indicating that the fix is being re−enabled. If no match is found before the first run or minimization is performed by the new script, the "state" information for the saved fix is discarded. See the doc pages for individual fixes for info on which ones can be restarted in this manner.

Bond interactions (angle, etc) that have been turned off by the fix shake or delete_bonds command will be written to a restart file as if they are turned on. This means they will need to be turned off again in a new run after the restart file is read.

Bonds that are broken (e.g. by a bond−breaking potential) are written to the restart file as broken bonds with a type of 0. Thus these bonds will still be broken when the restart file is read.

IMPORTANT NOTE: No other information is stored in the restart file. This means that an input script that reads a restart file should specify settings for quantities like timestep size, thermodynamic, neighbor list criteria including settings made via the neigh_modify comand, dump file output, geometric regions, etc.

Restrictions: none

Related commands:

read_data, write_restart, restart

Default: none

region command

Syntax:

```
region ID style args keyword arg ...
```

- ID = user−assigned name for the region
- style = *delete* or *block* or *cone* or *cylinder* or *plane* or *prism* or *sphere* or *union* or *intersect*

```
  delete = no args
    block args = xlo xhi ylo yhi zlo zhi
      xlo,xhi,ylo,yhi,zlo,zhi = bounds of block in all dimensions (distance units)
    cone args = dim c1 c2 radlo radhi lo hi
      dim = x or y or z = axis of cone
      c1,c2 = coords of cone axis in other 2 dimensions (distance units)
      radlo,radhi = cone radii at lo and hi end (distance units)
      lo,hi = bounds of cone in dim (distance units)
    cylinder args = dim c1 c2 radius lo hi
      dim = x or y or z = axis of cylinder
      c1,c2 = coords of cylinder axis in other 2 dimensions (distance units)
      radius = cylinder radius (distance units)
      lo,hi = bounds of cylinder in dim (distance units)
    plane args = px py pz nx ny nz
      px,py,pz = point on the plane (distance units)
      nx,ny,nz = direction normal to plane (distance units)
    prism args = xlo xhi ylo yhi zlo zhi xy xz yz
      xlo,xhi,ylo,yhi,zlo,zhi = bounds of untilted prism (distance units)
      xy = distance to tilt y in x direction (distance units)
      xz = distance to tilt z in x direction (distance units)
      yz = distance to tilt z in y direction (distance units)
    sphere args = x y z radius
      x,y,z = center of sphere (distance units)
      radius = radius of sphere (distance units)
    union args = N reg−ID1 reg−ID2 ...
      N = # of regions to follow, must be 2 or greater
      reg−ID1,reg−ID2, ... = IDs of regions to join together
    intersect args = N reg−ID1 reg−ID2 ...
      N = # of regions to follow, must be 2 or greater
      reg−ID1,reg−ID2, ... = IDs of regions to intersect
```

- zero or more keyword/arg pairs may be appended
- keyword = *side* or *units* or *move* or *rotate*

```
  side value = in or out
      in = the region is inside the specified geometry
      out = the region is outside the specified geometry
    units value = lattice or box
      lattice = the geometry is defined in lattice units
      box = the geometry is defined in simulation box units
    move args = v_x v_y v_z
      v_x,v_y,v_z = equal−style variables for x,y,z displacement of region over time
    rotate args = v_theta Px Py Pz Rx Ry Rz
      v_theta = equal−style variable for rotaton of region over time (in radians)
      Px,Py,Pz = origin for axis of rotation (distance units)
      Rx,Ry,Rz = axis of rotation vector
```

Examples:

```
region 1 block −3.0 5.0 INF 10.0 INF INF
```

```
region 2 sphere 0.0 0.0 0.0 5 side out
region void cylinder y 2 3 5 -5.0 EDGE units box
region 1 prism 0 10 0 10 0 10 2 0 0
region outside union 4 side1 side2 side3 side4
region 2 sphere 0.0 0.0 0.0 5 side out move v_left v_up NULL
```

Description:

This command defines a geometric region of space. Various other commands use regions. For example, the region can be filled with atoms via the create_atoms command. Or a bounding box around the region, can be used to define the simulation box via the create_box command. Or the atoms in the region can be identified as a group via the group command, or deleted via the delete_atoms command. Or the surface of the region can be used as a boundary wall via the fix wall/region command.

Commands which use regions typically test whether an atom's position is contained in the region or not. For this purpose, coordinates exactly on the region boundary are considered to be interior to the region. This means, for example, for a spherical region, an atom on the sphere surface would be part of the region if the sphere were defined with the *side in* keyword, but would not be part of the region if it were defined using the *side out* keyword. See more details on the *side* keyword below.

Normally, regions in LAMMPS are "static", meaning their geometric extent does not change with time. If the *move* or *rotate* keyword is used, as described below, the region becomes "dynamic", meaning it's location or orientation changes with time. This may be useful, for example, when thermostatting a region, via the compute temp/region command, or when the fix wall/region command uses a region surface as a bounding wall on particle motion, i.e. a rotating container.

The *delete* style removes the named region. Since there is little overhead to defining extra regions, there is normally no need to do this, unless you are defining and discarding large numbers of regions in your input script.

The lo/hi values for *block* or *cone* or *cylinder* or *prism* styles can be specified as EDGE or INF. EDGE means they extend all the way to the global simulation box boundary. Note that this is the current box boundary; if the box changes size during a simulation, the region does not. INF means a large negative or positive number (1.0e20), so it should encompass the simulation box even if it changes size. If a region is defined before the simulation box has been created (via create_box or read_data or read_restart commands), then an EDGE or INF parameter cannot be used. For a *prism* region, a non−zero tilt factor in any pair of dimensions cannot be used if both the lo/hi values in either of those dimensions are INF. E.g. if the xy tilt is non−zero, then xlo and xhi cannot both be INF, nor can ylo and yhi.

IMPORTANT NOTE: Regions in LAMMPS do not get wrapped across periodic boundaries, as specified by the boundary command. For example, a spherical region that is defined so that it overlaps a periodic boundary is not treated as 2 half−spheres, one on either side of the simulation box.

IMPORTANT NOTE: Regions in LAMMPS are always 3d geometric objects, regardless of whether the dimension of a simulation is 2d or 3d. Thus when using regions in a 2d simulation, you should be careful to define the region so that its intersection with the 2d x−y plane of the simulation has the 2d geometric extent you want.

For style *cone*, an axis−aligned cone is defined which is like a *cylinder* except that two different radii (one at each end) can be defined. Either of the radii (but not both) can be 0.0.

For style *cone* and *cylinder*, the c1,c2 params are coordinates in the 2 other dimensions besides the cylinder axis dimension. For dim = x, c1/c2 = y/z; for dim = y, c1/c2 = x/z; for dim = z, c1/c2 = x/y. Thus the third example above specifies a cylinder with its axis in the y−direction located at x = 2.0 and z = 3.0, with a radius of 5.0, and extending in the y−direction from −5.0 to the upper box boundary.

For style *plane*, a plane is defined which contain the point (px,py,pz) and has a normal vector (nx,ny,nz). The normal vector does not have to be of unit length. The "inside" of the plane is the half−space in the direction of the normal vector; see the discussion of the *side* option below.

For style *prism*, a parallelepiped is defined (it's too hard to spell parallelepiped in an input script!). The parallelepiped has its "origin" at (xlo,ylo,zlo) and is defined by 3 edge vectors starting from the origin given by A = (xhi−xlo,0,0); B = (xy,yhi−ylo,0); C = (xz,yz,zhi−zlo). *Xy,xz,yz* can be 0.0 or positive or negative values and are called "tilt factors" because they are the amount of displacement applied to faces of an originally orthogonal box to transform it into the parallelepiped.

A prism region that will be used with the create_box command to define a triclinic simulation box must have tilt factors (xy,xz,yz) that do not skew the box more than half the distance of corresponding the parallel box length. For example, if xlo = 2 and xhi = 12, then the x box length is 10 and the xy tilt factor must be between −5 and 5. Similarly, both xz and yz must be between −(xhi−xlo)/2 and +(yhi−ylo)/2. Note that this is not a limitation, since if the maximum tilt factor is 5 (as in this example), then configurations with tilt = ..., −15, −5, 5, 15, 25, ... are all geometrically equivalent.

See this section of the doc pages for a geometric description of triclinic boxes, as defined by LAMMPS, and how to transform these parameters to and from other commonly used triclinic representations.

The *union* style creates a region consisting of the volume of all the listed regions combined. The *intersect* style creates a region consisting of the volume that is common to all the listed regions.

The *side* keyword determines whether the region is considered to be inside or outside of the specified geometry. Using this keyword in conjunction with *union* and *intersect* regions, complex geometries can be built up. For example, if the interior of two spheres were each defined as regions, and a *union* style with *side* = out was constructed listing the region−IDs of the 2 spheres, the resulting region would be all the volume in the simulation box that was outside both of the spheres.

The *units* keyword determines the meaning of the distance units used to define the region for any argument above listed as having distance units. It also affects the scaling of the velocity vector specfied with the *vel* keyword, the amplitude vector specified with the *wiggle* keyword, and the rotation point specified with the *rotate* keyword, since they each involve a distance metric.

A *box* value selects standard distance units as defined by the units command, e.g. Angstroms for units = real or metal. A *lattice* value means the distance units are in lattice spacings. The lattice command must have been previously used to define the lattice spacings which are used as follows:

- For style *block*, the lattice spacing in dimension x is applied to xlo and xhi, similarly the spacings in dimensions y,z are applied to ylo/yhi and zlo/zhi.
- For style *cone*, the lattice spacing in argument *dim* is applied to lo and hi. The spacings in the two radial dimensions are applied to c1 and c2. The two cone radii are scaled by the lattice spacing in the dimension corresponding to c1.
- For style *cylinder*, the lattice spacing in argument *dim* is applied to lo and hi. The spacings in the two radial dimensions are applied to c1 and c2. The cylinder radius is scaled by the lattice spacing in the dimension corresponding to c1.
- For style *plane*, the lattice spacing in dimension x is applied to px and nx, similarly the spacings in dimensions y,z are applied to py/ny and pz/nz.
- For style *prism*, the lattice spacing in dimension x is applied to xlo and xhi, similarly for ylo/yhi and zlo/zhi. The lattice spacing in dimension x is applied to xy and xz, and the spacing in dimension y to yz.
- For style *sphere*, the lattice spacing in dimensions x,y,z are applied to the sphere center x,y,z. The spacing in dimension x is applied to the sphere radius.

If the *move* or *rotate* keywords are used, the region is "dynamic", meaning its location or orientation changes with time. These keywords cannot be used with a *union* or *intersect* style region. Instead, the keywords should be used to make the individual sub−regions of the *union* or *intersect* region dynamic. Normally, each sub−region should be "dynamic" in the same manner (e.g. rotate around the same point), though this is not a requirement.

The *move* keyword allows one or more equal−style variables to be used to specify the x,y,z displacement of the region, typically as a function of time. A variable is specified as v_name, where name is the variable name. Any of the three variables can be specified as NULL, in which case no displacement is calculated in that dimension.

Note that equal−style variables can specify formulas with various mathematical functions, and include thermo_style command keywords for the simulation box parameters and timestep and elapsed time. Thus it is easy to specify a region displacement that change as a function of time or spans consecutive runs in a continuous fashion. For the latter, see the *start* and *stop* keywords of the run command and the *elaplong* keyword of thermo_style custom for details.

For example, these commands would displace a region from its initial position, in the positive x direction, effectively at a constant velocity:

```
variable dx equal ramp(0,10)
region 2 sphere 10.0 10.0 0.0 5 move v_dx NULL NULL
```

Note that the initial displacemet is 0.0, though that is not required.

Either of these varaibles would "wiggle" the region back and forth in the y direction:

```
variable dy equal swiggle(0,5,100)
variable dysame equal 5*sin(2*PI*elaplong*dt/100)
region 2 sphere 10.0 10.0 0.0 5 move NULL v_dy NULL
```

The *rotate* keyword rotates the region around a rotation axis *R* = (Rx,Ry,Rz) that goes thru a point *P* = (Px,Py,Pz). The rotation angle is calculated, presumably as a function of time, by a variable specified as v_theta, where theta is the variable name. The variable should generate its result in radians. The direction of rotation for the region around the rotation axis is consistent with the right−hand rule: if your right−hand thumb points along *R*, then your fingers wrap around the axis in the direction of rotation.

The *move* and *rotate* keywords can be used together. In this case, the displacement specified by the *move* keyword is applied to the *P* point of the *rotate* keyword.

Restrictions:

A prism cannot be of 0.0 thickness in any dimension; use a small z thickness for 2d simulations. For 2d simulations, the xz and yz parameters must be 0.0.

Related commands:

lattice, create_atoms, delete_atoms, group

Default:

The option defaults are side = in, units = lattice, and no move or rotation.

replicate command

Syntax:

```
replicate nx ny nz
```

- nx,ny,nz = replication factors in each dimension

Examples:

```
replicate 2 3 2
```

Description:

Replicate the current simulation one or more times in each dimension. For example, replication factors of 2,2,2 will create a simulation with 8x as many atoms by doubling the simulation domain in each dimension. A replication factor of 1 in a dimension leaves the simulation domain unchanged.

All properties of the atoms are replicated, including their velocities, which may or may not be desirable. New atom IDs are assigned to new atoms, as are molecule IDs. Bonds and other topology interactions are created between pairs of new atoms as well as between old and new atoms. This is done by using the image flag for each atom to "unwrap" it out of the periodic box before replicating it. This means that molecular bonds you specify in the original data file that span the periodic box should be between two atoms with image flags that differ by 1. This will allow them to be unwrapped appropriately.

Restrictions:

A 2d simulation cannot be replicated in the z dimension.

If a simulation is non−periodic in a dimension, care should be used when replicating it in that dimension, as it may put atoms nearly on top of each other.

If the current simulation was read in from a restart file (before a run is performed), there can have been no fix information stored in the file for individual atoms. Similarly, no fixes can be defined at the time the replicate command is used that require vectors of atom information to be stored. This is because the replicate command does not know how to replicate that information for new atoms it creates.

Replicating a system that has rigid bodies (defined via the fix rigid command), either currently defined or that created the restart file which was read in before replicating, can cause problems if there is a bond between a pair of rigid bodies that straddle a periodic boundary. This is because the periodic image information for particles in the rigid bodies are set differently than for a non−rigid system and can result in a new bond being created that spans the periodic box. Thus you cannot use the replicate command in this scenario.

Related commands: none

Default: none

reset_timestep command

Syntax:

```
reset_timestep N
```

- N = timestep number

Examples:

```
reset_timestep 0
reset_timestep 4000000
```

Description:

Set the timestep counter to the specified value. This command normally comes after the timestep has been set by reading it in from a file or a previous simulation advanced the timestep.

The read_data and create_box commands set the timestep to 0; the read_restart command sets the timestep to the value it had when the restart file was written.

Restrictions: none

This command cannot be used when a dump file is defined via the dump command and has already been written to. It also cannot be used when a restart frequency has been set, and a restart file has already been written. This is because the changed timestep can mess up the planned timestep for the next file write. See the undump command or restart 0 command for info on how to turn off these definitions if necessary. New specifications for dump and restart files can be given after the reset_timestep command is used.

This command cannot be used when any fixes are defined that keep track of elapsed time to perform time−dependent operations. Examples include the "ave" fixes such as fix ave/spatial. Also fix dt/reset and fix deposit.

This command cannot be used when any dynamic regions are defined via the region command, which have time−dependent position and orientation.

There are other fixes which use the current timestep which may produce unexpected behavior, but LAMMPS allows them to be in place when resetting the timestep. For example, commands which thermostat the system, e.g. fix nvt, allow you to specify a target temperature which ramps from Tstart to Tstop which may persist over several runs. If you change the timestep, you may change the target temperature.

Resetting the timestep will clear the flags for computes that may have calculated some quantity from a previous run. This means that quantity cannot be accessed by a variable in between runs until a new run is performed. See the variable command for more details.

Related commands: none

Default: none

restart command

Syntax:

```
restart 0
restart N root
restart N file1 file2
```

- N = write a restart file every this many timesteps
- root = filename to which timestep # is appended
- file1,file2 = two full filenames, toggle between them when writing file

Examples:

```
restart 0
restart 1000 poly.restart
restart 1000 restart.*.equil
restart 10000 poly.%.1 poly.%.2
```

Description:

Write out a binary restart file every so many timesteps as a run proceeds. A value of 0 means do not write out restart files. Using one filename as an argument will create a series of filenames which include the timestep in the filename. Using two filenames will produce only 2 restart files. LAMMPS will toggle between the 2 names as it writes successive restart files.

Similar to dump files, the restart filename(s) can contain two wild−card characters. If a "*" appears in the filename, it is replaced with the current timestep value. This is only recognized when a single filename is used (not when toggling back and forth). Thus, the 3rd example above creates restart files as follows: restart.1000.equil, restart.2000.equil, etc. If a single filename is used with no "*", then the timestep value is appended. E.g. the 2nd example above creates restart files as follows: poly.restart.1000, poly.restart.2000, etc.

If a "%" character appears in the restart filename(s), then one file is written for each processor and the "%" character is replaced with the processor ID from 0 to P−1. An additional file with the "%" replaced by "base" is also written, which contains global information. For example, the files written on step 1000 for filename restart.% would be restart.base.1000, restart.0.1000, restart.1.1000, ..., restart.P−1.1000. This creates smaller files and can be a fast mode of output and subsequent input on parallel machines that support parallel I/O.

Restart files are written on timesteps that are a multiple of N but not on the first timestep of a run or minimization. A restart file is not written on the last timestep of a run unless it is a multiple of N. A restart file is written on the last timestep of a minimization if N > 0 and the minimization converges.

See the read_restart command for information about what is stored in a restart file.

Restart files can be read by a read_restart command to restart a simulation from a particular state. Because the file is binary (to enable exact restarts), it may not be readable on another machine. In this case, the restart2data program in the tools directory can be used to convert a restart file to an ASCII data file. Both the read_restart command and restart2data tool can read in a restart file that was written with the "%" character so that multiple files were created.

Restrictions: none

Related commands:

write_restart, read_restart

Default:

```
restart 0
```

run command

Syntax:

```
run N keyword values ...
```

- N = # of timesteps
- zero or more keyword/value pairs may be appended
- keyword = *upto* or *start* or *stop* or *pre* or *post* or *every*

```
  upto value = none
    start value = N1
      N1 = timestep at which 1st run started
    stop value = N2
      N2 = timestep at which last run will end
    pre value = no or yes
    post value = no or yes
    every values = M c1 c2 ...
      M = break the run into M-timestep segments and invoke one or more commands between each se
      c1,c2,...,cN = one or more LAMMPS commands, each enclosed in quotes
      c1 = NULL means no command will be invoked
```

Examples:

```
run 10000
run 1000000 upto
run 100 start 0 stop 1000
run 1000 pre no post yes
run 100000 start 0 stop 1000000 every 1000 "print 'Protein Rg = $r'"
run 100000 every 1000 NULL
```

Description:

Run or continue dynamics for a specified number of timesteps.

When the run style is *respa*, N refers to outer loop (largest) timesteps.

A value of N = 0 is acceptable; only the thermodynamics of the system are computed and printed without taking a timestep.

The *upto* keyword means to perform a run starting at the current timestep up to the specified timestep. E.g. if the current timestep is 10,000 and "run 100000 upto" is used, then an additional 90,000 timesteps will be run. This can be useful for very long runs on a machine that allocates chunks of time and terminate your job when time is exceeded. If you need to restart your script multiple times (reading in the last restart file), you can keep restarting your script with the same run command until the simulation finally completes.

The *start* or *stop* keywords can be used if multiple runs are being performed and you want a fix command that changes some value over time (e.g. temperature) to make the change across the entire set of runs and not just a single run. See the doc page for individual fixes to see which ones can be used with the *start/stop* keywords.

For example, consider this fix followed by 10 run commands:

```
fix          1 all nvt 200.0 300.0 1.0
run          1000 start 0 stop 10000
```

```
run          1000 start 0 stop 10000
...
run          1000 start 0 stop 10000
```

The NVT fix ramps the target temperature from 200.0 to 300.0 during a run. If the run commands did not have the start/stop keywords (just "run 1000"), then the temperature would ramp from 200.0 to 300.0 during the 1000 steps of each run. With the start/stop keywords, the ramping takes place over the 10000 steps of all runs together.

The *pre* and *post* keywords can be used to streamline the setup, clean−up, and associated output to the screen that happens before and after a run. This can be useful if you wish to do many short runs in succession (e.g. LAMMPS is being called as a library which is doing other computations between successive short LAMMPS runs).

By default (pre and post = yes), LAMMPS creates neighbor lists, computes forces, and imposes fix constraints before every run. And after every run it gathers and prints timings statistics. If a run is just a continuation of a previous run (i.e. no settings are changed), the initial computation is not necessary; the old neighbor list is still valid as are the forces. So if *pre* is specified as "no" then the initial setup is skipped, except for printing thermodynamic info. Note that if *pre* is set to "no" for the very 1st run LAMMPS performs, then it is overridden, since the initial setup computations must be done.

IMPORTANT NOTE: If your input script changes settings between 2 runs (e.g. adds a fix or dump or compute or changes a neighbor list parameter), then the initial setup must be performed. LAMMPS does not check for this, but it would be an error to use the *pre no* option in this case.

If *post* is specified as "no", the full timing summary is skipped; only a one−line summary timing is printed.

The *every* keyword provides a means of breaking a LAMMPS run into a series of shorter runs. Optionally, one or more LAMMPS commands (c1, c2, ..., cN) will be executed in between the short runs. If used, the *every* keyword must be the last keyword, since it has a variable number of arguments. Each of the trailing arguments is a single LAMMPS command, and each command should be enclosed in quotes, so that the entire command will be treated as a single argument. This will also prevent any variables in the command from being evaluated until it is executed multiple times during the run. Note that if a command itself needs one of its arguments quoted (e.g. the print command), then you can use a combination of single and double quotes, as in the example above or below.

The *every* keyword is a means to avoid listing a long series of runs and interleaving commands in your input script. For example, a print command could be invoked or a fix could be redefined, e.g. to reset a thermostat temperature. Or this could be useful for invoking a command you have added to LAMMPS that wraps some other code (e.g. as a library) to perform a computation periodically during a long LAMMPS run. See this section of the documentation for info about how to add new commands to LAMMPS. See this section of the documentation for ideas about how to couple LAMMPS to other codes.

With the *every* option, N total steps are simulated, in shorter runs of M steps each. After each M−length run, the specified commands are invoked. If only a single command is specified as NULL, then no command is invoked. Thus these lines:

```
variable q equal x[100]
run 6000 every 2000 "print Coord = $q"
```

are the equivalent of:

```
variable q equal x[100]
run 2000
print Coord = $q
run 2000
print Coord = $q
run 2000
```

```
print Coord = $q
```

which does 3 runs of 2000 steps and prints the x−coordinate of a particular atom between runs. Note that the variable "$q" will be evaluated afresh each time the print command is executed.

Note that by using the line continuation character "the run every command can be spread across many lines, though it is still a single command:

```
run 100000 every 1000 &  "print 'Minimum value = $a'" &  "print 'Maximum value = $b'" &  "print 'Tem
```

If the *pre* and *post* options are set to "no" when used with the *every* keyword, then the 1st run will do the full setup and the last run will print the full timing summary, but these operations will be skipped for intermediate runs.

Restrictions:

The number of specified timesteps N must fit in a signed 32−bit integer, so you are limited to slightly more than 2 billion steps (2^31) in a single run. However, you can perform successive runs to run a simulation for any number of steps (ok, up to 2^63 steps).

Related commands:

minimize, run_style, temper

Default:

The option defaults are start = the current timestep, stop = current timestep + N, pre = yes, and post = yes.

run_style command

Syntax:

```
run_style style args
```

- style = *verlet* or *respa*

```
  verlet args = none
    respa args = N n1 n2 ... keyword values ...
      N = # of levels of rRESPA
      n1, n2, ... = loop factor between rRESPA levels (N−1 values)
      zero or more keyword/value pairings may be appended to the loop factors
      keyword = bond or angle or dihedral or improper or
        pair or inner or middle or outer or kspace
        bond value = M
          M = which level (1−N) to compute bond forces in
        angle value = M
          M = which level (1−N) to compute angle forces in
        dihedral value = M
          M = which level (1−N) to compute dihedral forces in
        improper value = M
          M = which level (1−N) to compute improper forces in
        pair value = M
          M = which level (1−N) to compute pair forces in
        inner values = M cut1 cut2
          M = which level (1−N) to compute pair inner forces in
          cut1 = inner cutoff between pair inner and
                 pair middle or outer  (distance units)
          cut2 = outer cutoff between pair inner and
                 pair middle or outer  (distance units)
        middle values = M cut1 cut2
          M = which level (1−N) to compute pair middle forces in
          cut1 = inner cutoff between pair middle and pair outer (distance units)
          cut2 = outer cutoff between pair middle and pair outer (distance units)
        outer value = M
          M = which level (1−N) to compute pair outer forces in
        kspace value = M
          M = which level (1−N) to compute kspace forces in
```

Examples:

```
run_style verlet
run_style respa 4 2 2 2 bond 1 dihedral 2 pair 3 kspace 4
run_style respa 4 2 2 2 bond 1 dihedral 2 inner 3 5.0 6.0 outer 4 kspace 4
```

Description:

Choose the style of time integrator used for molecular dynamics simulations performed by LAMMPS.

The *verlet* style is a velocity−Verlet integrator.

The *respa* style implements the rRESPA multi−timescale integrator (Tuckerman) with N hierarchical levels, where level 1 is the innermost loop (shortest timestep) and level N is the outermost loop (largest timestep). The loop factor arguments specify what the looping factor is between levels. N1 specifies the number of iterations of level 1 for a single iteration of level 2, N2 is the iterations of level 2 per iteration of level 3, etc. N−1 looping

parameters must be specified.

The timestep command sets the timestep for the outermost rRESPA level. Thus if the example command above for a 4−level rRESPA had an outer timestep of 4.0 fmsec, the inner timestep would be 8x smaller or 0.5 fmsec. All other LAMMPS commands that specify number of timesteps (e.g. neigh_modify parameters, dump every N timesteps, etc) refer to the outermost timesteps.

The rRESPA keywords enable you to specify at what level of the hierarchy various forces will be computed. If not specified, the defaults are that bond forces are computed at level 1 (innermost loop), angle forces are computed where bond forces are, dihedral forces are computed where angle forces are, improper forces are computed where dihedral forces are, pair forces are computed at the outermost level, and kspace forces are computed where pair forces are. The inner, middle, outer forces have no defaults.

The *inner* and *middle* keywords take additional arguments for cutoffs that are used by the pairwise force computations. If the 2 cutoffs for *inner* are 5.0 and 6.0, this means that all pairs up to 6.0 apart are computed by the inner force. Those between 5.0 and 6.0 have their force go ramped to 0.0 so the overlap with the next regime (middle or outer) is smooth. The next regime (middle or outer) will compute forces for all pairs from 5.0 outward, with those from 5.0 to 6.0 having their value ramped in an inverse manner.

Only some pair potentials support the use of the *inner* and *middle* and *outer* keywords. If not, only the *pair* keyword can be used with that pair style, meaning all pairwise forces are computed at the same rRESPA level. See the doc pages for individual pair styles for details.

When using rRESPA (or for any MD simulation) care must be taken to choose a timestep size(s) that insures the Hamiltonian for the chosen ensemble is conserved. For the constant NVE ensemble, total energy must be conserved. Unfortunately, it is difficult to know *a priori* how well energy will be conserved, and a fairly long test simulation (~10 ps) is usually necessary in order to verify that no long−term drift in energy occurs with the trial set of parameters.

With that caveat, a few rules−of−thumb may be useful in selecting *respa* settings. The following applies mostly to biomolecular simulations using the CHARMM or a similar all−atom force field, but the concepts are adaptable to other problems. Without SHAKE, bonds involving hydrogen atoms exhibit high−frequency vibrations and require a timestep on the order of 0.5 fmsec in order to conserve energy. The relatively inexpensive force computations for the bonds, angles, impropers, and dihedrals can be computed on this innermost 0.5 fmsec step. The outermost timestep cannot be greater than 4.0 fmsec without risking energy drift. Smooth switching of forces between the levels of the rRESPA hierarchy is also necessary to avoid drift, and a 1−2 angstrom "healing distance" (the distance between the outer and inner cutoffs) works reasonably well. We thus recommend the following settings for use of the *respa* style without SHAKE in biomolecular simulations:

```
timestep  4.0
run_style respa 4 2 2 2 inner 2 4.5 6.0 middle 3 8.0 10.0 outer 4
```

With these settings, users can expect good energy conservation and roughly a 2.5 fold speedup over the *verlet* style with a 0.5 fmsec timestep.

If SHAKE is used with the *respa* style, time reversibility is lost, but substantially longer time steps can be achieved. For biomolecular simulations using the CHARMM or similar all−atom force field, bonds involving hydrogen atoms exhibit high frequency vibrations and require a time step on the order of 0.5 fmsec in order to conserve energy. These high frequency modes also limit the outer time step sizes since the modes are coupled. It is therefore desirable to use SHAKE with respa in order to freeze out these high frequency motions and increase the size of the time steps in the respa hierarchy. The following settings can be used for biomolecular simulations with SHAKE and rRESPA:

```
fix             2 all shake 0.000001 500 0 m 1.0 a 1
timestep        4.0
run_style       respa 2 2 inner 1 4.0 5.0 outer 2
```

With these settings, users can expect good energy conservation and roughly a 1.5 fold speedup over the *verlet* style with SHAKE and a 2.0 fmsec timestep.

For non−biomolecular simulations, the *respa* style can be advantageous if there is a clear separation of time scales − fast and slow modes in the simulation. Even a LJ system can benefit from rRESPA if the interactions are divided by the inner, middle and outer keywords. A 2−fold or more speedup can be obtained while maintaining good energy conservation. In real units, for a pure LJ fluid at liquid density, with a sigma of 3.0 angstroms, and epsilon of 0.1 Kcal/mol, the following settings seem to work well:

```
timestep  36.0
run_style respa 3 3 4 inner 1 3.0 4.0 middle 2 6.0 7.0 outer 3
```

Restrictions:

Whenever using rRESPA, the user should experiment with trade−offs in speed and accuracy for their system, and verify that they are conserving energy to adequate precision.

Related commands:

timestep, run

Default:

```
run_style verlet
```

(Tuckerman) Tuckerman, Berne and Martyna, J Chem Phys, 97, p 1990 (1992).

set command

Syntax:

```
set style ID keyword values ...
```

- style = *atom* or *type* or *mol* or *group* or *region*
- ID = atom ID range or type range or mol ID range or group ID or region ID
- one or more keyword/value pairs may be appended
- keyword = *type* or *type/fraction* or *mol* or *x* or *y* or *z* or *charge* or *dipole* or *dipole/random* or *quat* or *quat/random* or *diameter* or *shape* or *mass* or *density* or *volume* or *image* or *bond* or *angle* or *dihedral* or *improper*

```
type value = atom type
  type/fraction values = type fraction seed
    type = new atom type
    fraction = fraction of selected atoms to set to new atom type
    seed = random # seed (positive integer)
  mol value = molecule ID
  x,y,z value = atom coordinate (distance units)
  charge value = atomic charge (charge units)
  dipole values = x y z
    x,y,z = orientation of dipole moment vector
  dipole/random value = seed Dlen
    seed = random # seed (positive integer) for dipole moment orientations
    Dlen = magnitude of dipole moment (dipole units)
  quat values = a b c theta
    a,b,c = unit vector to rotate particle around via right-hand rule
    theta = rotation angle in degrees
  quat/random value = seed
    seed = random # seed (positive integer) for quaternion orientations
  diameter value = diameter of spherical particle (distance units)
  shape value = Sx Sy Sz
    Sx,Sy,Sz = 3 diameters of ellipsoid (distance units)
  mass value = per-atom mass (mass units)
  density value = particle density for sphere or ellipsoid (mass/distance^3 units)
  volume value = particle volume for Peridynamic particle (distance^3 units)
  image nx ny nz
    nx,ny,nz = which periodic image of the simulation box the atom is in
  bond value = bond type for all bonds between selected atoms
  angle value = angle type for all angles between selected atoms
  dihedral value = dihedral type for all dihedrals between selected atoms
  improper value = improper type for all impropers between selected atoms
```

Examples:

```
set group solvent type 2
set group solvent type/fraction 2 0.5 12393
set group edge bond 4
set region half charge 0.5
set type 3 charge 0.5
set type 1*3 charge 0.5
set atom 100*200 x 0.5 y 1.0
set atom 1492 type 3
```

Description:

Set one or more properties of one or more atoms. Since atom properties are initially assigned by the read_data, read_restart or create_atoms commands, this command changes those assignments. This can be useful for overriding the default values assigned by the create_atoms command (e.g. charge = 0.0). It can be useful for altering pairwise and molecular force interactions, since force−field coefficients are defined in terms of types. It can be used to change the labeling of atoms by atom type or molecule ID when they are output in dump files. It can be useful for debugging purposes; i.e. positioning an atom at a precise location to compute subsequent forces or energy.

The style *atom* selects all the atoms in a range of atom IDs. The style *type* selects all the atoms in a range of types. The style *mol* selects all the atoms in a range of molecule IDs.

In each of the range cases, a single value can be specified, or a wildcard asterisk can be used to specify a range of values. This takes the form "*" or "*n" or "n*" or "m*n". For example, for the style *type*, if N = the number of atom types, then an asterisk with no numeric values means all types from 1 to N. A leading asterisk means all types from 1 to n (inclusive). A trailing asterisk means all types from n to N (inclusive). A middle asterisk means all types from m to n (inclusive). Note that the loweest value for the wildcard is 1, not 0, so you cannot not use this form to select atoms with molecule ID = 0, for example.

The style *group* selects all the atoms in the specified group. The style *region* selects all the atoms in the specified geometric region. See the group and region commands for details of how to specify a group or region.

Keyword *type* sets the atom type for all selected atoms. The specified value must be from 1 to ntypes, where ntypes was set by the create_box command or the *atom types* field in the header of the data file read by the read_data command.

Keyword *type/fraction* sets the atom type for a fraction of the selected atoms. The actual number of atoms changed is not guaranteed to be exactly the requested fraction, but should be statistically close. Random numbers are used in such a way that a particular atom is changed or not changed, regardless of how many processors are being used.

Keyword *mol* sets the molecule ID for all selected atoms. The atom style being used must support the use of molecule IDs.

Keywords *x*, *y*, *z*, and *charge* set the coordinates or charge of all selected atoms. For *charge*, the atom style being used must support the use of atomic charge.

Keyword *dipole* uses the specified x,y,z values as components of a vector to set as the orientation of the dipole moment vectors of the selected atoms. The magnitude of the dipole moment is set by the length of this orientation vector.

Keyword *dipole/random* randomizes the orientation of the dipole moment vectors of the selected atoms and sets the magnitude of each to the specified *Dlen* value. For 2d systems, the z component of the orientation is set to 0.0. Random numbers are used in such a way that the orientation of a particular atom is the same, regardless of how many processors are being used.

Keyword *quat* uses the specified values to create a quaternion (4−vector) that represents the orientation of the selected atoms. Note that particles defined by atom_style ellipsoid have 3 shape parameters. The 3 values must be non−zero for each particle set by this command. They are used to specify the aspect ratios of an ellipsoidal particle, which is oriented by default with its x−axis along the simulation box's x−axis, and similarly for y and z. If this body is rotated (via the right−hand rule) by an angle theta around a unit rotation vector (a,b,c), then the quaternion that represents its new orientation is given by (cos(theta/2), a*sin(theta/2), b*sin(theta/2), c*sin(theta/2)). The theta and a,b,c values are the arguments to the *quat* keyword. LAMMPS normalizes the

quaternion in case (a,b,c) was not specified as a unit vector. For 2d systems, the a,b,c values are ignored, since a rotation vector of (0,0,1) is the only valid choice.

Keyword *quat/random* randomizes the orientation of the quaternion of the selected atoms. Random numbers are used in such a way that the orientation of a particular atom is the same, regardless of how many processors are being used. For 2d systems, only orientations in the xy plane are generated. As with keyword *quat*, the 3 shape values must be non−zero for each particle set by this command.

Keyword *diameter* sets the size of the selected atoms. The particles must be finite−size spheres as defined by the atom_style sphere command. The diameter of a particle can be set to 0.0, which means they will be treated as point particles. Note that this command does not adjust the particle mass, even if it was defined with a density, e.g. via the read_data command.

Keyword *shape* sets the size and shape of the selected atoms. The particles must be aspherical ellipsoids as defined by the atom_style ellipsoid command. The *Sx*, *Sy*, *Sz* settings are the 3 diameters of the ellipsoid in each direction. All 3 can be set to the same value, which means the ellipsoid is effectively a sphere. They can also all be set to 0.0 which means the particle will be treated as a point particle. Note that this command does not adjust the particle mass, even if it was defined with a density, e.g. via the read_data command.

Keyword *mass* sets the mass of all selected particles. The particles must have a per−atom mass attribute, as defined by the atom_style command. See the "mass" command for how to set mass values on a per−type basis.

Keyword *density* sets the mass of all selected particles. The particles must have a per−atom mass attribute, as defined by the atom_style command. See the "mass" command for how to set mass values on a per−type basis. If the atom has a radius attribute (see atom_style sphere) and its radius is non−zero, its mass is set from the density and particle volume. The same is true if the atom has a shape attribute (see atom_style ellipsoid) and its 3 shape parameters are non−zero. Otherwise the mass is set to the density value directly.

Keyword *volume* sets the volume of all selected particles. Currently, only the atom_style peri command defines particles with a volume attribute. Note that this command does not adjust the particle mass.

Keyword *image* sets which image of the simulation box the atom is considered to be in. An image of 0 means it is inside the box as defined. A value of 2 means add 2 box lengths to get the true value. A value of −1 means subtract 1 box length to get the true value. LAMMPS updates these flags as atoms cross periodic boundaries during the simulation. The flags can be output with atom snapshots via the dump command. If a value of NULL is specified for any of nx,ny,nz, then the current image value for that dimension is unchanged. For non−periodic dimensions only a value of 0 can be specified.

This command can be useful after a system has been equilibrated and atoms have diffused one or more box lengths in various directions. This command can then reset the image values for atoms so that they are effectively inside the simulation box, e.g if a diffusion coefficient is about to be measured via the compute msd command. Care should be taken not to reset the image flags of two atoms in a bond to the same value if the bond straddles a periodic boundary (rather they should be different by +/− 1). This will not affect the dynamics of a simulation, but may mess up analysis of the trajectories if a LAMMPS diagnostic or your own analysis relies on the image flags to unwrap a molecule which straddles the periodic box.

Keywords *bond*, *angle*, *dihedral*, and *improper*, set the bond type (angle type, etc) of all bonds (angles, etc) of selected atoms to the specified value from 1 to nbondtypes (nangletypes, etc). All atoms in a particular bond (angle, etc) must be selected atoms in order for the change to be made. The value of nbondtype (nangletypes, etc) was set by the *bond types* (*angle types*, etc) field in the header of the data file read by the read_data command.

Restrictions:

You cannot set an atom attribute (e.g. *mol* or *q* or *volume*) if the atom_style does not have that attribute.

This command requires inter−processor communication to coordinate the setting of bond types (angle types, etc). This means that your system must be ready to perform a simulation before using one of these keywords (force fields set, atom mass set, etc). This is not necessary for other keywords.

Using the *region* style with the bond (angle, etc) keywords can give unpredictable results if there are bonds (angles, etc) that straddle periodic boundaries. This is because the region may only extend up to the boundary and partner atoms in the bond (angle, etc) may have coordinates outside the simulation box if they are ghost atoms.

Related commands:

create_box, create_atoms, read_data

Default: none

shell command

Syntax:

```
shell cmd args
```

- cmd = *cd* or *mkdir* or *mv* or *rm* or *rmdir* or arbitrary command

```
cd arg = dir
    dir = directory to change to
  mkdir args = dir1 dir2 ...
    dir1,dir2 = one or more directories to create
  mv args = old new
    old = old filename
    new = new filename
  rm args = file1 file2 ...
    file1,file2 = one or more filenames to delete
  rmdir args = dir1 dir2 ...
    dir1,dir2 = one or more directories to delete
  anything else is passed as a command to the shell for direct execution
```

Examples:

```
shell cd sub1
shell cd ..
shell mkdir tmp1 tmp2 tmp3
shell rmdir tmp1
shell mv log.lammps hold/log.1
shell rm TMP/file1 TMP/file2
shell my_setup file1 10 file2
shell my_post_process 100 dump.out
```

Description:

Execute a shell command. A few simple file−based shell commands are supported directly, in Unix−style syntax. Any command not listed above is passed as−is to the C−library system() call, which invokes the command in a shell.

This is means to invoke other commands from your input script. For example, you can move files around in preparation for the next section of the input script. Or you can run a program that pre−processes data for input into LAMMPS. Or you can run a program that post−processes LAMMPS output data.

With the exception of *cd*, all commands, including ones invoked via a system() call, are executed by only a single processor, so that files/directories are not being manipulated by multiple processors.

The *cd* cmd executes the Unix "cd" command to change the working directory. All subsequent LAMMPS commands that read/write files will use the new directory. All processors execute this command.

The *mkdir* cmd executes the Unix "mkdir" command to create one or more directories.

The *mv* cmd executes the Unix "mv" command to rename a file and/or move it to a new directory.

The *rm* cmd executes the Unix "rm" command to remove one or more files.

The *rmdir* cmd executes the Unix "rmdir" command to remove one or more directories. A directory must be empty to be successfully removed.

Any other cmd is passed as−is to the shell along with its arguments as one string, invoked by the C−library system() call. For example, these lines in your input script:

```
variable n equal 10
variable foo string file2
shell my_setup file1 $n ${foo}
```

would be the same as invoking

```
% my_setup file1 10 file2
```

from a command−line prompt. The executable program "my_setup" is run with 3 arguments: file1 10 file2.

Restrictions:

LAMMPS does not detect errors or print warnings when any of these commands execute. E.g. if the specified directory does not exist, executing the *cd* command will silently do nothing.

Related commands: none

Default: none

special_bonds command

Syntax:

```
special_bonds keyword values ...
```

- one or more keyword/value pairs may be appended
- keyword = *amber* or *charmm* or *dreiding* or *fene* or *lj/coul* or *lj* or *coul* or *angle* or *dihedral* or *extra*

```
amber values = none
  charmm values = none
  dreiding values = none
  fene values = none
  lj/coul values = w1,w2,w3
    w1,w2,w3 = weights (0.0 to 1.0) on pairwise Lennard−Jones and Coulombic interactions
  lj values = w1,w2,w3
    w1,w2,w3 = weights (0.0 to 1.0) on pairwise Lennard−Jones interactions
  coul values = w1,w2,w3
    w1,w2,w3 = weights (0.0 to 1.0) on pairwise Coulombic interactions
  angle value = yes or no
  dihedral value = yes or no
  extra value = N
    N = number of extra 1−2,1−3,1−4 interactions to save space for
```

Examples:

```
special_bonds amber
special_bonds charmm
special_bonds fene dihedral no
special_bonds lj/coul 0.0 0.0 0.5 angle yes dihedral yes
special_bonds lj 0.0 0.0 0.5 coul 0.0 0.0 0.0 dihedral yes
special_bonds lj/coul 0 1 1 extra 2
```

Description:

Set weighting coefficients for pairwise energy and force contributions between pairs of atoms that are also permanently bonded to each other, either directly or via one or two intermediate bonds. These weighting factors are used by nearly all pair styles in LAMMPS that compute simple pairwise interactions. Permanent bonds between atoms are specified by defining the bond topology in the data file read by the read_data command. Typically a bond_style command is also used to define a bond potential. The rationale for using these weighting factors is that the interaction between a pair of bonded atoms is all (or mostly) specified by the bond, angle, dihedral potentials, and thus the non−bonded Lennard−Jones or Coulombic interaction between the pair of atoms should be excluded (or reduced by a weighting factor).

IMPORTANT NOTE: These weighting factors are NOT used by pair styles that compute many−body interactions, since the "bonds" that result from such interactions are not permanent, but are created and broken dynamically as atom conformations change. Examples of pair styles in this category are EAM, MEAM, Stillinger−Weber, Tersoff, COMB, AIREBO, and ReaxFF. In fact, it generally makes no sense to define permanent bonds between atoms that interact via these potentials, though such bonds may exist elsewhere in your system, e.g. when using the pair_style hybrid command. Thus LAMMPS ignores special_bonds settings when manybody potentials are calculated.

The Coulomb factors are applied to any Coulomb (charge interaction) term that the potential calculates. The LJ factors are applied to the remaining terms that the potential calculates, whether they represent LJ interactions or

not. The weighting factors are a scaling pre−factor on the energy and force between the pair of atoms. A value of 1.0 means include the full interaction; a value of 0.0 means exclude it completely.

The 1st of the 3 coefficients (LJ or Coulombic) is the weighting factor on 1−2 atom pairs, which are pairs of atoms directly bonded to each other. The 2nd coefficient is the weighting factor on 1−3 atom pairs which are those separated by 2 bonds (e.g. the two H atoms in a water molecule). The 3rd coefficient is the weighting factor on 1−4 atom pairs which are those separated by 3 bonds (e.g. the 1st and 4th atoms in a dihedral interaction). Thus if the 1−2 coefficient is set to 0.0, then the pairwise interaction is effectively turned off for all pairs of atoms bonded to each other. If it is set to 1.0, then that interaction will be at full strength.

IMPORTANT NOTE: For purposes of computing weighted pairwise interactions, 1−3 and 1−4 interactions are not defined from the list of angles or dihedrals used by the simulation. Rather, they are inferred topologically from the set of bonds specified when the simulation is defined from a data or restart file (see read_data or read_restart commands). Thus the set of 1−2,1−3,1−4 interactions that the weights apply to is the same whether angle and dihedral potentials are computed or not, and remains the same even if bonds are constrained, or turned off, or removed during a simulation.

The two exceptions to this rule are (a) if the *angle* or *dihedral* keywords are set to *yes* (see below), or (b) if the delete_bonds command is used with the *special* option that recomputes the 1−2,1−3,1−4 topologies after bonds are deleted; see the delete_bonds command for more details.

The *amber* keyword sets the 3 coefficients to 0.0, 0.0, 0.5 for LJ interactions and to 0.0, 0.0, 0.8333 for Coulombic interactions, which is the default for a commonly used version of the AMBER force field, where the last value is really 5/6. See (Cornell) for a description of the AMBER force field.

The *charmm* keyword sets the 3 coefficients to 0.0, 0.0, 0.0 for both LJ and Coulombic interactions, which is the default for a commonly used version of the CHARMM force field. Note that in pair styles *lj/charmm/coul/charmm* and *lj/charmm/coul/long* the 1−4 coefficients are defined explicitly, and these pairwise contributions are computed as part of the charmm dihedral style − see the pair_coeff and dihedral_style commands for more information. See (MacKerell) for a description of the CHARMM force field.

The *dreiding* keyword sets the 3 coefficients to 0.0, 0.0, 1.0 for both LJ and Coulombic interactions, which is the default for the Dreiding force field, as discussed in (Mayo).

The *fene* keyword sets the 3 coefficients to 0.0, 1.0, 1.0 for both LJ and Coulombic interactions, which is consistent with a coarse−grained polymer model with FENE bonds. See (Kremer) for a description of FENE bonds.

The *lj/coul*, *lj*, and *coul* keywords allow the 3 coefficients to be set explicitly. The *lj/coul* keyword sets both the LJ and Coulombic coefficients to the same 3 values. The *lj* and *coul* keywords only set either the LJ or Coulombic coefficients. Use both of them if you wish to set the LJ coefficients to different values than the Coulombic coefficients.

The *angle* keyword allows the 1−3 weighting factor to be ignored for individual atom pairs if they are not listed as the first and last atoms in any angle defined in the simulation or as 1,3 or 2,4 atoms in any dihedral defined in the simulation. For example, imagine the 1−3 weighting factor is set to 0.5 and you have a linear molecule with 4 atoms and bonds as follows: 1−2−3−4. If your data file defines 1−2−3 as an angle, but does not define 2−3−4 as an angle or 1−2−3−4 as a dihedral, then the pairwise interaction between atoms 1 and 3 will always be weighted by 0.5, but different force fields use different rules for weighting the pairwise interaction between atoms 2 and 4. If the *angle* keyword is specified as *yes*, then the pairwise interaction between atoms 2 and 4 will be unaffected (full weighting of 1.0). If the *angle* keyword is specified as *no* which is the default, then the 2,4 interaction will also be weighted by 0.5.

The *dihedral* keyword allows the 1−4 weighting factor to be ignored for individual atom pairs if they are not listed as the first and last atoms in any dihedral defined in the simulation. For example, imagine the 1−4 weighting factor is set to 0.5 and you have a linear molecule with 5 atoms and bonds as follows: 1−2−3−4−5. If your data file defines 1−2−3−4 as a dihedral, but does not define 2−3−4−5 as a dihedral, then the pairwise interaction between atoms 1 and 4 will always be weighted by 0.5, but different force fields use different rules for weighting the pairwise interaction between atoms 2 and 5. If the *dihedral* keyword is specified as *yes*, then the pairwise interaction between atoms 2 and 5 will be unaffected (full weighting of 1.0). If the *dihedral* keyword is specified as *no* which is the default, then the 2,5 interaction will also be weighted by 0.5.

The *extra* keyword is used when additional bonds will be created during a simulation run, e.g. by the fix bond/create command. A list of 1−2,1−3,1−4 neighbors for each atom is calculated and stored by LAMMPS. If new bonds are created, the list needs to grow. Using the *extra* keyword leaves empty space in the list for N additional bonds to be added. If you do not do this, you may get an error when bonds are added.

Restrictions: none

Related commands:

delete_bonds, fix bond/create

Default:

All 3 Lennard−Jones and 3 Coulobmic weighting coefficients = 0.0, angle = no, dihedral = no, and extra = 0.

(Cornell) Cornell, Cieplak, Bayly, Gould, Merz, Ferguson, Spellmeyer, Fox, Caldwell, Kollman, JACS 117, 5179−5197 (1995).

(Kremer) Kremer, Grest, J Chem Phys, 92, 5057 (1990).

(MacKerell) MacKerell, Bashford, Bellott, Dunbrack, Evanseck, Field, Fischer, Gao, Guo, Ha, et al, J Phys Chem, 102, 3586 (1998).

(Mayo) Mayo, Olfason, Goddard III, J Phys Chem, 94, 8897−8909 (1990).

suffix command

Syntax:

```
suffix style
```

- style = *off* or *on* or *opt* or *gpu* or *cuda*

Examples:

```
suffix off
suffix on
suffix gpu
```

Description:

This command allows you to use variants of various styles if they exist. In that respect it operates the same as the −suffix command−line switch. It also has options to turn off/on any suffix setting made via the command line.

The specified style can be *opt* or *gpu* or *cuda*. These refer to optional packages that LAMMPS can be built with, as described in this section of the manual. The "opt" style corrsponds to the OPT package, the "gpu" style to the GPU package, and the "cuda" style to the USER−CUDA package.

These are the variants these packages provide:

- OPT = a handful of pair styles, cache−optimized for faster CPU performance
- GPU = a handful of pair styles and the PPPM kspace_style, optimized to run on one or more GPUs or multicore CPU/GPU nodes
- USER−CUDA = a collection of atom, pair, fix, compute, and intergrate styles, optimized to run on one or more NVIDIA GPUs

As an example, all of the packages provide a pair_style lj/cut variant, with style names lj/cut/opt or lj/cut/gpu or lj/cut/cuda. A variant styles can be specified explicitly in your input script, e.g. pair_style lj/cut/gpu. If the suffix command is used with the appropriate style, you do not need to modify your input script. The specified suffix (opt,gpu,cuda) is automatically appended whenever your input script command creates a new atom, pair, fix, compute, or run style. If the variant version does not exist, the standard version is created.

If the specified style is *off*, then any previously specified suffix is temporarily disabled, whether it was specified by a command−line switch or a previous suffix command. If the specified style is *on*, a disabled suffix is turned back on. The use of these 2 commands lets your input script use a standard LAMMPS style (i.e. a non−accelerated variant), which can be useful for testing or benchmarking purposes. Of course this is also possible by not using any suffix commands, and explictly appending or not appending the suffix to the relevant commands in your input script.

Restrictions: none

Related commands:

Command−line switch −suffix

Default: none

tad command

Syntax:

```
tad N t_event T_lo T_hi delta tmax compute−ID seed keyword value ...
```

- N = # of timesteps to run (not including dephasing/quenching)
- t_event = timestep interval between event checks
- T_lo = temperature at which event times are desired
- T_hi = temperature at which MD simulation is performed
- delta = desired confidence level for stopping criterion
- tmax = reciprocal of lowest expected preexponential factor (time units)
- compute−ID = ID of the compute used for event detection
- zero or more keyword/value pairs may be appended
- keyword = *min* or *neb* or *min_style* or *neb_style* or *neb_log*

```
  min values = etol ftol maxiter maxeval
      etol = stopping tolerance for energy (energy units)
      ftol = stopping tolerance for force (force units)
      maxiter = max iterations of minimize
      maxeval = max number of force/energy evaluations
    neb values = ftol N1 N2 Nevery
      etol = stopping tolerance for energy (energy units)
      ftol = stopping tolerance for force (force units)
      N1 = max # of iterations (timesteps) to run initial NEB
      N2 = max # of iterations (timesteps) to run barrier−climbing NEB
      Nevery = print NEB statistics every this many timesteps
    min_style value = cg or hftn or sd or quickmin or fire
    neb_style value = quickmin or fire
    neb_log value = file where NEB statistics are printed
```

Examples:

```
tad 2000 50 1800 2300 0.01 0.01 event 54985
tad 2000 50 1800 2300 0.01 0.01 event 54985 &    min 1e−05 1e−05 100 100 &    neb 0.0 0.01 200 200 2
```

Description:

Run a temperature accelerated dynamics (TAD) simulation. This method requires two or more partitions to perform NEB transition state searches.

TAD is described in this paper by Art Voter. It is a method that uses accelerated dynamics at an elevated temperature to generate results at a specified lower temperature. A good overview of accelerated dynamics methods for such systems is given in this review paper from the same group. In general, these methods assume that the long−time dynamics is dominated by infrequent events i.e. the system is is confined to low energy basins for long periods, punctuated by brief, randomly−occurring transitions to adjacent basins. TAD is suitable for infrequent−event systems, where in addition, the transition kinetics are well−approximated by harmonic transition state theory (hTST). In hTST, the temperature dependence of transition rates follows the Arrhenius relation. As a consequence a set of event times generated in a high−temperature simulation can be mapped to a set of much longer estimated times in the low−temperature system. However, because this mapping involves the energy barrier of the transition event, which is different for each event, the first event at the high temperature may not be the earliest event at the low temperature. TAD handles this by first generating a set of possible events from the current basin. After each event, the simulation is reflected backwards into the current basin. This is repeated until

the stopping criterion is satisfied, at which point the event with the earliest low−temperature occurrence time is selected. The stopping criterion is that the confidence measure be greater than 1−*delta*. The confidence measure is the probability that no earlier low−temperature event will occur at some later time in the high−temperature simulation. hTST provides an lower bound for this probability, based on the user−specified minimum pre−exponential factor (reciprocal of *tmax*).

In order to estimate the energy barrier for each event, the TAD method invokes the NEB method. Each NEB replica runs on a partition of processors. The current NEB implementation in LAMMPS restricts you to having exactly one processor per replica. For more information, see the documentation for the neb command. In the current LAMMPS implementation of TAD, all the non−NEB TAD operations are performed on the first partition, while the other partitions remain idle. See this section of the manual for further discussion of multi−replica simulations.

A TAD run has several stages, which are repeated each time an event is performed. The logic for a TAD run is as follows:

```
while (time remains):
  while (time <tstop):
    until (event occurs):
      run dynamics for t_event steps
      quench
    run neb calculation using all replicas
    compute tlo from energy barrier
    update earliest event
    update tstop
    reflect back into current basin
  execute earliest event
```

Before this outer loop begins, the initial potential energy basin is identified by quenching (an energy minimization, see below) the initial state and storing the resulting coordinates for reference.

Inside the inner loop, dynamics is run continuously according to whatever integrator has been specified by the user, stopping every *t_event* steps to check if a transition event has occurred. This check is performed by quenching the system and comparing the resulting atom coordinates to the coordinates from the previous basin.

A quench is an energy minimization and is performed by whichever algorithm has been defined by the *min* and *min_style* keywords or their default values. Note that typically, you do not need to perform a highly−converged minimization to detect a transition event.

The event check is performed by a compute with the specified *compute−ID*. Currently there is only one compute that works with the TAD commmand, which is the compute event/displace command. Other event−checking computes may be added. Compute event/displace checks whether any atom in the compute group has moved further than a specified threshold distance. If so, an "event" has occurred.

The neb calculation is similar to that invoked by the neb command, except that the final state is generated internally, instead of being read in from a file. The TAD implementation provides default values for the NEB settings, which can be overridden using the *neb* and *neb_style* keywords.

A key aspect of the TAD method is setting the stopping criterion appropriately. If this criterion is too conservative, then many events must be generated before one is finally executed. Conversely, if this criterion is too aggressive, high−entropy high−barrier events will be over−sampled, while low−entropy low−barrier events will be under−sampled. If the lowest pre−exponential factor is known fairly accurately, then it can be used to estimate *tmax*, and the value of *delta* can be set to the desired confidence level e.g. *delta* = 0.05 corresponds to 95% confidence. However, for systems where the dynamics are not well characterized (the most common case), it

will be necessary to experiment with the values of *delta* and *tmax* to get a good trade−off between accuracy and performance.

A second key aspect is the choice of *t_hi*. A larger value greatly increases the rate at which new events are generated. However, too large a value introduces errors due to anharmonicity (not accounted for within hTST). Once again, for any given system, experimentation is necessary to determine the best value of *t_hi*.

Five kinds of output can be generated during a TAD run: event statistics, NEB statistics, thermodynamic output by each replica, dump files, and restart files.

Event statistics are printed to the screen and master log.lammps file each time an event is executed. The quantities are the timestep, CPU time, global event number *N*, local event number *M*, event status, energy barrier, time margin, *t_lo* and *delt_lo*. The timestep is the usual LAMMPS timestep, which corresponds to the high−temperature time at which the event was detected, in units of timestep. The CPU time is the total processor time since the start of the TAD run. The global event number *N* is a counter that increments with each executed event. The local event number *M* is a counter that resets to zero upon entering each new basin. The event status is *E* when an event is executed, and is *D* for an event that is detected, while *DF* is for a detected event that is also the earliest (first) event at the low temperature.

The time margin is the ratio of the high temperature time in the current basin to the stopping time. This last number can be used to judge whether the stopping time is too short or too long (see above).

t_lo is the low−temperature event time when the current basin was entered, in units of timestep. del*t_lo* is the time of each detected event, measured relative to *t_lo*. *delt_lo* is equal to the high−temperature time since entering the current basin, scaled by an exponential factor that depends on the hi/lo temperature ratio and the energy barrier for that event.

On lines for executed events, with status *E*, the global event number is incremented by one, and the timestep, local event number, energy barrier, *t_lo*, and *delt_lo* match the last event with status *DF* in the immediately preceding block of detected events.

The NEB statistics are written to the file specified by the *neb_log* keyword. If the keyword value is "none", then no NEB statistics are printed out. The statistics are written every *Nevery* timesteps. See the neb command for a full description of the NEB statistics. When invoked from TAD, NEB statistics are never printed to the screen.

Because the NEB calculation must run on multiple partitions, LAMMPS produces additional screen and log files for each partition, e.g. log.lammps.0, log.lammps.1, etc. For the TAD command, these contain the thermodynamic output of each NEB replica. In addition, the log file for the first partition, log.lammps.0, will contain thermodynamic output from short runs and minimizations corresponding to the dynamics and quench operations, as well as a line for each new detected event, as described above.

After the TAD command completes, timing statistics for the TAD run are printed in each replica's log file, giving a breakdown of how much CPU time was spent in each stage (NEB, dynamics, quenching, etc).

Any dump files defined in the input script will be written to during a TAD run at timesteps when an event is executed. This means the the requested dump frequency in the dump command is ignored. There will be one dump file (per dump command) created for all partitions. The atom coordinates of the dump snapshot are those of the minimum energy configuration resulting from quenching following the executed event. The timesteps written into the dump files correspond to the timestep at which the event occurred and NOT the clock. A dump snapshot corresponding to the initial minimum state used for event detection is written to the dump file at the beginning of each TAD run.

If the restart command is used, a single restart file for all the partitions is generated, which allows a TAD run to be continued by a new input script in the usual manner. The restart file is generated after an event is executed. The restart file contains a snapshot of the system in the new quenched state, including the event number and the low−temperature time. The restart frequency specified in the restart command is interpreted differently when performing a TAD run. It does not mean the timestep interval between restart files. Instead it means an event interval for executed events. Thus a frequency of 1 means write a restart file every time an event is executed. A frequency of 10 means write a restart file every 10th executed event. When an input script reads a restart file from a previous TAD run, the new script can be run on a different number of replicas or processors.

Note that within a single state, the dynamics will typically temporarily continue beyond the event that is ultimately chosen, until the stopping criterionis satisfied. When the event is eventually executed, the timestep counter is reset to the value when the event was detected. Similarly, after each quench and NEB minimization, the timestep counter is reset to the value at the start of the minimization. This means that the timesteps listed in the replica log files do not always increase monotonically. However, the timestep values printed to the master log file, dump files, and restart files are always monotonically increasing.

Restrictions:

This command can only be used if LAMMPS was built with the "replica" package. See the Making LAMMPS section for more info on packages.

N setting must be integer multiple of *t_event*.

Runs restarted from restart files written during a TAD run will only produce identical results if the user−specified integrator supports exact restarts. So fix nvt will produce an exact restart, but fix langevin will not.

This command cannot be used when any fixes are defined that keep track of elapsed time to perform time−dependent operations. Examples include the "ave" fixes such as fix ave/spatial. Also fix dt/reset and fix deposit.

Related commands:

compute event/displace, min_modify, min_style, run_style, minimize, temper, neb, prd

Default:

The option defaults are *min* = 0.1 0.1 40 50, *neb* = 0.01 100 100 10, *min_style* = *cg*, *neb_style* = *quickmin*, and *neb_log* = "none"

(Voter) Sørensen and Voter, J Chem Phys, 112, 9599 (2000)

(Voter2) Voter, Montalenti, Germann, Annual Review of Materials Research 32, 321 (2002).

temper command

Syntax:

```
temper N M temp fix-ID seed1 seed2 index
```

- N = total # of timesteps to run
- M = attempt a tempering swap every this many steps
- temp = initial temperature for this ensemble
- fix−ID = ID of the fix that will control temperature during the run
- seed1 = random # seed used to decide on adjacent temperature to partner with
- seed2 = random # seed for Boltzmann factor in Metropolis swap
- index = which temperature (0 to N−1) I am simulating (optional)

Examples:

```
temper 100000 100 $t tempfix 0 58728
temper 40000 100 $t tempfix 0 32285 $w
```

Description:

Run a parallel tempering or replica exchange simulation using multiple replicas (ensembles) of a system. Two or more replicas must be used.

Each replica runs on a partition of one or more processors. Processor partitions are defined at run−time using the −partition command−line switch; see this section of the manual. Note that if you have MPI installed, you can run a multi−replica simulation with more replicas (partitions) than you have physical processors, e.g you can run a 10−replica simulation on one or two processors. You will simply not get the performance speed−up you would see with one or more physical processors per replica. See this section of the manual for further discussion.

Each replica's temperature is controlled at a different value by a fix with *fix−ID* that controls temperature. Possible fix styles are nvt, temp/berendsen, langevin and temp/rescale. The desired temperature is specified by *temp*, which is typically a variable previously set in the input script, so that each partition is assigned a different temperature. See the variable command for more details. For example:

```
variable t world 300.0 310.0 320.0 330.0
fix myfix all nvt $t $t 100.0
temper 100000 100 $t myfix 3847 58382
```

would define 4 temperatures, and assign one of them to the thermostat used by each replica, and to the temper command.

As the tempering simulation runs for *N* timesteps, a temperature swap between adjacent ensembles will be attempted every *M* timesteps. If *seed1* is 0, then the swap attempts will alternate between odd and even pairings. If *seed1* is non−zero then it is used as a seed in a random number generator to randomly choose an odd or even pairing each time. Each attempted swap of temperatures is either accepted or rejected based on a Boltzmann−weighted Metropolis criterion which uses *seed2* in the random number generator.

As a tempering run proceeds, multiple log files and screen output files are created, one per replica. By default these files are named log.lammps.M and screen.M where M is the replica number from 0 to N−1, with N = # of replicas. See the section on command−line switches for info on how to change these names.

The main screen and log file (log.lammps) will list information about which temperature is assigned to each replica at each thermodynamic output timestep. E.g. for a simulation with 16 replicas:

```
Running on 16 partitions of processors
Step T0 T1 T2 T3 T4 T5 T6 T7 T8 T9 T10 T11 T12 T13 T14 T15
0    0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
500  1 0 3 2 5 4 6 7 8 9 10 11 12 13 14 15
1000 2 0 4 1 5 3 6 7 8 9 10 11 12 14 13 15
1500 2 1 4 0 5 3 6 7 9 8 10 11 12 14 13 15
2000 2 1 3 0 6 4 5 7 10 8 9 11 12 14 13 15
2500 2 1 3 0 6 4 5 7 11 8 9 10 12 14 13 15
...
```

The column headings T0 to TN−1 mean which temperature is currently assigned to the replica 0 to N−1. Thus the columns represent replicas and the value in each column is its temperature (also numbered 0 to N−1). For example, a 0 in the 4th column (column T3, step 2500) means that the 4th replica is assigned temperature 0, i.e. the lowest temperature. You can verify this time sequence of temperature assignments for the Nth replica by comparing the Nth column of screen output to the thermodynamic data in the corresponding log.lammps.N or screen.N files as time proceeds.

The last argument *index* in the temper command is optional and is used when restarting a tempering run from a set of restart files (one for each replica) which had previously swapped to new temperatures. The *index* value (from 0 to N−1, where N is the # of replicas) identifies which temperature the replica was simulating on the timestep the restart files were written. Obviously, this argument must be a variable so that each partition has the correct value. Set the variable to the *N* values listed in the log file for the previous run for the replica temperatures at that timestep. For example if the log file listed the following for a simulation with 5 replicas:

```
500000 2 4 0 1 3
```

then a setting of

```
variable w proc 2 4 0 1 3
```

would be used to restart the run with a tempering command like the example above with $w as the last argument.

Restrictions:

This command can only be used if LAMMPS was built with the "replica" package. See the Making LAMMPS section for more info on packages.

Related commands:

variable, prd, neb

Default: none

thermo command

Syntax:

```
thermo N
```

- N = output thermodynamics every N timesteps

Examples:

```
thermo 100
```

Description:

Compute and print thermodynamic info (e.g. temperature, energy, pressure) on timesteps that are a multiple of N and at the beginning and end of a simulation. A value of 0 will only print thermodynamics at the beginning and end.

The content and format of what is printed is controlled by the thermo_style and thermo_modify commands.

The timesteps on which thermo output is written can also be controlled by a variable. See the thermo_modify every command.

Restrictions: none

Related commands:

thermo_style, thermo_modify

Default:

```
thermo 0
```

thermo_modify command

Syntax:

```
thermo_modify keyword value ...
```

- one or more keyword/value pairs may be listed
- keyword = *lost* or *norm* or *flush* or *line* or *format* or *temp* or *press* or *every*

```
lost value = error or warn or ignore
  norm value = yes or no
  flush value = yes or no
  line value = one or multi
  format values = int string or float string or M string
    M = integer from 1 to N, where N = # of quantities being printed
    string = C−style format string
  temp value = compute ID that calculates a temperature
  press value = compute ID that calculates a pressure
  every value = v_name
    v_name = an equal−style variable name
```

Examples:

```
thermo_modify lost ignore flush yes
thermo_modify temp myTemp format 3 %15.8g
thermo_modify line multi format float %g
```

Description:

Set options for how thermodynamic information is computed and printed by LAMMPS.

IMPORTANT NOTE: These options apply to the currently defined thermo style. When you specify a thermo_style command, all thermodynamic settings are restored to their default values, including those previously reset by a thermo_modify command. Thus if your input script specifies a thermo_style command, you should use the thermo_modify command after it.

The *lost* keyword determines whether LAMMPS checks for lost atoms each time it computes thermodynamics and what it does if atoms are lost. An atom can be "lost" if it moves across a non−periodic simulation box boundary or if it moves more than a box length outside the simulation domain (or more than a processor sub−domain length) before reneighboring occurs. The latter case is typically due to bad dynamics, e.g. too large a timestep or huge forces and velocities. If the value is *ignore*, LAMMPS does not check for lost atoms. If the value is *error* or *warn*, LAMMPS checks and either issues an error or warning. The code will exit with an error and continue with a warning. A warning will only be issued once, the first time an atom is lost. This can be a useful debugging option.

The *norm* keyword determines whether various thermodynamic output values are normalized by the number of atoms or not, depending on whether it is set to *yes* or *no*. Different unit styles have different defaults for this setting (see below). Even if *norm* is set to *yes*, a value is only normalized if it is an "extensive" quantity, meaning that it scales with the number of atoms in the system. For the thermo keywords described by the doc page for the thermo_style command, all energy−related keywords are extensive, such as *pe* or *ebond* or *enthalpy*. Other keywords such as *temp* or *press* are "intensive" meaning their value is independent (in a statistical sense) of the number of atoms in the system and thus are never normalized. For thermodynamic output values extracted from fixes and computes in a thermo_style custom command, the doc page for the individual fix or compute lists

whether the value is "extensive" or "intensive" and thus whether it is normalized. Thermodynamic output values calculated by a variable formula are assumed to be "intensive" and thus are never normalized. You can always include a divide by the number of atoms in the variable formula if this is not the case.

The *flush* keyword invokes a flush operation after thermodynamic info is written to the log file. This insures the output in that file is current (no buffering by the OS), even if LAMMPS halts before the simulation completes.

The *line* keyword determines whether thermodynamics will be printed as a series of numeric values on one line or in a multi−line format with 3 quantities with text strings per line and a dashed−line header containing the timestep and CPU time. This modify option overrides the *one* and *multi* thermo_style settings.

The *format* keyword sets the numeric format of individual printed quantities. The *int* and *float* keywords set the format for all integer or floating−point quantities printed. The setting with a numeric value M (e.g. format 5 %10.4g) sets the format of the Mth value printed in each output line, e.g. the 5th column of output in this case. If the format for a specific column has been set, it will take precedent over the *int* or *float* setting.

IMPORTANT NOTE: The thermo output values *step* and *atoms* are stored internally as 8−byte unsigned integers, rather than the usual 4−byte signed integers. When specifying the "format int" keyword you can use a "%d"−style format identifier in the format string and LAMMPS will convert this to the corresponding "%lu" form when it is applied to those keywords. However, when specifying the "format M string" keyword for *step* and *natoms*, you should specify a string appropriate for an 8−byte unsigned integer, e.g. one with "%lu".

The *temp* keyword is used to determine how thermodynamic temperature is calculated, which is used by all thermo quantities that require a temperature ("temp", "press", "ke", "etotal", "enthalpy", "pxx", etc). The specified compute ID must have been previously defined by the user via the compute command and it must be a style of compute that calculates a temperature. As described in the thermo_style command, thermo output uses a default compute for temperature with ID = *thermo_temp*. This option allows the user to override the default.

The *press* keyword is used to determine how thermodynamic pressure is calculated, which is used by all thermo quantities that require a pressure ("press", "enthalpy", "pxx", etc). The specified compute ID must have been previously defined by the user via the compute command and it must be a style of compute that calculates a pressure. As described in the thermo_style command, thermo output uses a default compute for pressure with ID = *thermo_press*. This option allows the user to override the default.

IMPORTANT NOTE: If both the *temp* and *press* keywords are used in a single thermo_modify command (or in two separate commands), then the order in which the keywords are specified is important. Note that a pressure compute defines its own temperature compute as an argument when it is specified. The *temp* keyword will override this (for the pressure compute being used by thermodynamics), but only if the *temp* keyword comes after the *press* keyword. If the *temp* keyword comes before the *press* keyword, then the new pressure compute specified by the *press* keyword will be unaffected by the *temp* setting.

The *every* keyword allows a variable to be specified which will determine which timesteps thermodynamic output is generated. It must be an equal−style variable, and is specified as v_name, where name is the variable name. The variable is evaluated at the beginning of a run to determine the next timestep at which a dump snapshot will be written out. On that timestep, the variable will be evaluated again to determine the next timestep, etc. Thus the variable should return timestep values. See the stagger() and logfreq() math functions for equal−style variables, as examples of useful functions to use in this context. Other similar math functions could easily be added as options for equal−style variables. In addition, thermodynamic output will always occur on the first and last timestep of each run.

For example, the following commands will output thermodynamic info at timesteps 0,10,20,30,100,200,300,1000,2000,etc:

```
variable        s equal logfreq(10,3,10)
thermo_modify   1 every v_s
```

Note that the *every* keyword overrides the output frequency setting made by the thermo command, by setting it to 0. If the thermo command is later used to set the output frequency to a non−zero value, then the variable setting of the thermo_modify every command will be overridden.

Restrictions: none

Related commands:

thermo, thermo_style

Default:

The option defaults are lost = error, norm = yes for unit style of *lj*, norm = no for unit style of *real* and *metal*, flush = no, temp/press = compute IDs defined by thermo_style, every = non−variable setting provided by the thermo command.

The defaults for the line and format options depend on the thermo style. For styles "one" and "custom", the line and format defaults are "one", "%8d", and "%12.8g". For style "multi", the line and format defaults are "multi", "%8d", and "%14.4f".

thermo_style command

Syntax:

```
thermo_style style args
```

- style = *one* or *multi* or *custom*
- args = list of arguments for a particular style

```
one args = none
  multi args = none
  custom args = list of attributes
    possible attributes = step, elapsed, elaplong, dt, cpu, tpcpu, spcpu,
                          atoms, temp, press, pe, ke, etotal, enthalpy,
                          evdwl, ecoul, epair, ebond, eangle, edihed, eimp,
                          emol, elong, etail,
                          vol, lx, ly, lz, xlo, xhi, ylo, yhi, zlo, zhi,
                          xy, xz, yz, xlat, ylat, zlat,
                          pxx, pyy, pzz, pxy, pxz, pyz,
                          fmax, fnorm,
                          cella, cellb, cellc, cellalpha, cellbeta, cellgamma,
                          c_ID, c_ID[I], c_ID[I][J],
                          f_ID, f_ID[I], f_ID[I][J],
                          v_name
      step = timestep
      elapsed = timesteps since start of this run
      elaplong = timesteps since start of initial run in a series of runs
      dt = timestep size
      cpu = elapsed CPU time in seconds
      tpcpu = time per CPU second
      spcpu = timesteps per CPU second
      atoms = # of atoms
      temp = temperature
      press = pressure
      pe = total potential energy
      ke = kinetic energy
      etotal = total energy (pe + ke)
      enthalpy = enthalpy (etotal + press*vol)
      evdwl = VanderWaal pairwise energy
      ecoul = Coulombic pairwise energy
      epair = pairwise energy (evdwl + ecoul + elong + etail)
      ebond = bond energy
      eangle = angle energy
      edihed = dihedral energy
      eimp = improper energy
      emol = molecular energy (ebond + eangle + edihed + eimp)
      elong = long-range kspace energy
      etail = VanderWaal energy long-range tail correction
      vol = volume
      lx,ly,lz = box lengths in x,y,z
      xlo,xhi,ylo,yhi,zlo,zhi = box boundaries
      xy,xz,yz = box tilt for triclinic (non-orthogonal) simulation boxes
      xlat,ylat,zlat = lattice spacings as calculated by lattice command
      pxx,pyy,pzz,pxy,pxz,pyz = 6 components of pressure tensor
      fmax = max component of force on any atom in any dimension
      fnorm = length of force vector for all atoms
      cella,cellb,cellc = periodic cell lattice constants a,b,c
      cellalpha, cellbeta, cellgamma = periodic cell angles alpha,beta,gamma
      c_ID = global scalar value calculated by a compute with ID
      c_ID[I] = Ith component of global vector calculated by a compute with ID
```

```
c_ID[I][J] = I,J component of global array calculated by a compute with ID
f_ID = global scalar value calculated by a fix with ID
f_ID[I] = Ith component of global vector calculated by a fix with ID
f_ID[I][J] = I,J component of global array calculated by a fix with ID
v_name = scalar value calculated by an equal-style variable with name
```

Examples:

```
thermo_style multi
thermo_style custom step temp pe etotal press vol
thermo_style custom step temp etotal c_myTemp v_abc
```

Description:

Set the style and content for printing thermodynamic data to the screen and log file.

Style *one* prints a one-line summary of thermodynamic info that is the equivalent of "thermo_style custom step temp epair emol etotal press". The line contains only numeric values.

Style *multi* prints a multiple-line listing of thermodynamic info that is the equivalent of "thermo_style custom etotal ke temp pe ebond eangle edihed eimp evdwl ecoul elong press". The listing contains numeric values and a string ID for each quantity.

Style *custom* is the most general setting and allows you to specify which of the keywords listed above you want printed on each thermodynamic timestep. Note that the keywords c_ID, f_ID, v_name are references to computes, fixes, and equal-style variables that have been defined elsewhere in the input script or can even be new styles which users have added to LAMMPS (see the Section_modify section of the documentation). Thus the *custom* style provides a flexible means of outputting essentially any desired quantity as a simulation proceeds.

All styles except *custom* have *vol* appended to their list of outputs if the simulation box volume changes during the simulation.

The values printed by the various keywords are instantaneous values, calculated on the current timestep. Time-averaged quantities, which include values from previous timesteps, can be output by using the f_ID keyword and accessing a fix that does time-averaging such as the fix ave/time command.

Options invoked by the thermo_modify command can be used to set the one- or multi-line format of the print-out, the normalization of thermodynamic output (total values versus per-atom values for extensive quantities (ones which scale with the number of atoms in the system), and the numeric precision of each printed value.

IMPORTANT NOTE: When you use a "thermo_style" command, all thermodynamic settings are restored to their default values, including those previously set by a thermo_modify command. Thus if your input script specifies a thermo_style command, you should use the thermo_modify command after it.

Several of the thermodynamic quantities require a temperature to be computed: "temp", "press", "ke", "etotal", "enthalpy", "pxx", etc. By default this is done by using a *temperature* compute which is created when LAMMPS starts up, as if this command had been issued:

```
compute thermo_temp all temp
```

See the compute temp command for details. Note that the ID of this compute is *thermo_temp* and the group is *all*. You can change the attributes of this temperature (e.g. its degrees-of-freedom) via the compute_modify command. Alternatively, you can directly assign a new compute (that calculates temperature) which you have

defined, to be used for calculating any thermodynamic quantity that requires a temperature. This is done via the thermo_modify command.

Several of the thermodynamic quantities require a pressure to be computed: "press", "enthalpy", "pxx", etc. By default this is done by using a *pressure* compute which is created when LAMMPS starts up, as if this command had been issued:

```
compute thermo_press all pressure thermo_temp
```

See the compute pressure command for details. Note that the ID of this compute is *thermo_press* and the group is *all*. You can change the attributes of this pressure via the compute_modify command. Alternatively, you can directly assign a new compute (that calculates pressure) which you have defined, to be used for calculating any thermodynamic quantity that requires a pressure. This is done via the thermo_modify command.

Several of the thermodynamic quantities require a potential energy to be computed: "pe", "etotal", "ebond", etc. This is done by using a *pe* compute which is created when LAMMPS starts up, as if this command had been issued:

```
compute thermo_pe all pe
```

See the compute pe command for details. Note that the ID of this compute is *thermo_pe* and the group is *all*. You can change the attributes of this potential energy via the compute_modify command.

The kinetic energy of the system *ke* is inferred from the temperature of the system with 1/2 Kb T of energy for each degree of freedom. Thus, using different compute commands for calculating temperature, via the thermo_modify temp command, may yield different kinetic energies, since different computes that calculate temperature can subtract out different non−thermal components of velocity and/or include different degrees of freedom (translational, rotational, etc).

The potential energy of the system *pe* will include contributions from fixes if the fix_modify thermo option is set for a fix that calculates such a contribution. For example, the fix wall/lj93 fix calculates the energy of atoms interacting with the wall. See the doc pages for "individual fixes" to see which ones contribute.

A long−range tail correction *etail* for the VanderWaal pairwise energy will be non−zero only if the pair_modify tail option is turned on. The *etail* contribution is included in *evdwl*, *pe*, and *etotal*, and the corresponding tail correction to the pressure is included in *press* and *pxx*, *pyy*, etc.

The *step*, *elapsed*, and *elaplong* keywords refer to timestep count. *Step* is the current timestep, or iteration count when a minimization is being performed. *Elapsed* is the number of timesteps elapsed since the beginning of this run. *Elaplong* is the number of timesteps elapsed since the beginning of an initial run in a series of runs. See the *start* and *stop* keywords for the run for info on how to invoke a series of runs that keep track of an initial starting time. If these keywords are not used, then *elapsed* and *elaplong* are the same value.

The *cpu* keyword is elapsed CPU seconds since the beginning of this run. The *tpcpu* and *spcpu* keywords are measures of how fast your simulation is currently running. The *tpcpu* keyword is simulation time per CPU second, where simulation time is in time units. E.g. for metal units, the *tpcpu* value would be picoseconds per CPU second. The *spcpu* keyword is the number of timesteps per CPU second. Both quantities are on−the−fly metrics, measured relative to the last time they were invoked. Thus if you are printing out thermodyamic output every 100 timesteps, the two keywords will continually output the time and timestep rate for the last 100 steps. The *tpcpu* keyword does not attempt to track any changes in timestep size, e.g. due to using the fix dt/reset command.

The *fmax* and *fnorm* keywords are useful for monitoring the progress of an energy minimization. The *fmax* keyword calculates the maximum force in any dimension on any atom in the system, or the infinity−norm of the force vector for the system. The *fnorm* keyword calculates the 2−norm or length of the force vector.

The keywords *cella*, *cellb*, *cellc*, *cellalpha*, *cellbeta*, *cellgamma*, correspond to the usual crystallographic quantities that define the periodic unit cell of a crystal. See this section of the doc pages for a geometric description of triclinic periodic cells, including a precise defintion of these quantities in terms of the internal LAMMPS cell dimensions *lx*, *ly*, *lz*, *yz*, *xz*, *xy*,

---

The *c_ID* and *c_ID[I]* and *c_ID[I][J]* keywords allow global values calculated by a compute to be output. As discussed on the compute doc page, computes can calculate global, per−atom, or local values. Only global values can be referenced by this command. However, per−atom compute values can be referenced in a variable and the variable referenced by thermo_style custom, as discussed below.

The ID in the keyword should be replaced by the actual ID of a compute that has been defined elsewhere in the input script. See the compute command for details. If the compute calculates a global scalar, vector, or array, then the keyword formats with 0, 1, or 2 brackets will reference a scalar value from the compute.

Note that some computes calculate "intensive" global quantities like temperature; others calculate "extensive" global quantities like kinetic energy that are summed over all atoms in the compute group. Intensive quantities are printed directly without normalization by thermo_style custom. Extensive quantities may be normalized by the total number of atoms in the simulation (NOT the number of atoms in the compute group) when output, depending on the thermo_modify norm option being used.

The *f_ID* and *f_ID[I]* and *f_ID[I][J]* keywords allow global values calculated by a fix to be output. As discussed on the fix doc page, fixes can calculate global, per−atom, or local values. Only global values can be referenced by this command. However, per−atom fix values can be referenced in a variable and the variable referenced by thermo_style custom, as discussed below.

The ID in the keyword should be replaced by the actual ID of a fix that has been defined elsewhere in the input script. See the fix command for details. If the fix calculates a global scalar, vector, or array, then the keyword formats with 0, 1, or 2 brackets will reference a scalar value from the fix.

Note that some fixes calculate "intensive" global quantities like timestep size; others calculate "extensive" global quantities like energy that are summed over all atoms in the fix group. Intensive quantities are printed directly without normalization by thermo_style custom. Extensive quantities may be normalized by the total number of atoms in the simulation (NOT the number of atoms in the fix group) when output, depending on the thermo_modify norm option being used.

The *v_name* keyword allow the current value of a variable to be output. The name in the keyword should be replaced by the variable name that has been defined elsewhere in the input script. Only equal−style variables can be referenced. See the variable command for details. Variables of style *equal* can reference per−atom properties or thermodynamic keywords, or they can invoke other computes, fixes, or variables when evaluated, so this is a very general means of creating thermodynamic output.

See this section for information on how to add new compute and fix styles to LAMMPS to calculate quantities that can then be referenced with these keywords to generate thermodynamic output.

Restrictions:

This command must come after the simulation box is defined by a read_data, read_restart, or create_box command.

Related commands:

thermo, thermo_modify, fix_modify, compute temp, compute pressure

Default:

```
thermo_style one
```

timestep command

Syntax:

```
timestep dt
```

- dt = timestep size (time units)

Examples:

```
timestep 2.0
timestep 0.003
```

Description:

Set the timestep size for subsequent molecular dynamics simulations. See the units command for a discussion of time units. The default value for the timestep also depends on the choice of units for the simulation; see the default values below.

When the run style is *respa*, dt is the timestep for the outer loop (largest) timestep.

Restrictions: none

Related commands:

fix dt/reset, run, run_style respa, units

Default:

timestep = 0.005 tau for units = lj
timestep = 1.0 fmsec for units = real
timestep = 0.001 psec for units = metal
timestep = 1.0e−8 sec (10 nsec) for units = si or cgs

uncompute command

Syntax:

```
uncompute compute-ID
```

- compute-ID = ID of a previously defined compute

Examples:

```
uncompute 2
uncompute lower-boundary
```

Description:

Delete a compute that was previously defined with a compute command. This also wipes out any additional changes made to the compute via the compute_modify command.

Restrictions: none

Related commands:

compute

Default: none

undump command

Syntax:

```
undump dump-ID
```

- dump-ID = ID of previously defined dump

Examples:

```
undump mine
undump 2
```

Description:

Turn off a previously defined dump so that it is no longer active. This closes the file associated with the dump.

Restrictions: none

Related commands:

dump

Default: none

unfix command

Syntax:

```
unfix fix-ID
```

- fix−ID = ID of a previously defined fix

Examples:

```
unfix 2
unfix lower-boundary
```

Description:

Delete a fix that was previously defined with a fix command. This also wipes out any additional changes made to the fix via the fix_modify command.

Restrictions: none

Related commands:

fix

Default: none

units command

Syntax:

```
units style
```

- style = *lj* or *real* or *metal* or *si* or *cgs* or *electron*

Examples:

```
units metal
units lj
```

Description:

This command sets the style of units used for a simulation. It determines the units of all quantities specified in the input script and data file, as well as quantities output to the screen, log file, and dump files. Typically, this command is used at the very beginning of an input script.

For all units except *lj*, LAMMPS uses physical constants from www.physics.nist.gov. For the definition of Kcal in real units, LAMMPS uses the thermochemical calorie = 4.184 J.

For style *lj*, all quantities are unitless. Without loss of generality, LAMMPS sets the fundamental quantities mass, sigma, epsilon, and the Boltzmann constant = 1. The masses, distances, energies you specify are multiples of these fundamental values. The formulas relating the reduced or unitless quantity (with an asterisk) to the same quantity with units is also given. Thus you can use the mass &sigma &epsilon values for a specific material and convert the results from a unitless LJ simulation into physical quantities.

- mass = mass or m
- distance = sigma, where x* = x / sigma
- time = tau, where tau = t* = t (epsilon / m / sigma^2)^1/2
- energy = epsilon, where E* = E / epsilon
- velocity = sigma/tau, where v* = v tau / sigma
- force = epsilon/sigma, where f* = f sigma / epsilon
- torque = epsilon, where t* = t / epsilon
- temperature = reduced LJ temperature, where T* = T Kb / epsilon
- pressure = reduced LJ pressure, where P* = P sigma^3 / epsilon
- dynamic viscosity = reduced LJ viscosity, where eta* = eta sigma^3 / epsilon / tau
- charge = reduced LJ charge, where q* = q / (4 pi perm0 sigma epsilon)^1/2
- dipole = reduced LJ dipole, moment where *mu = mu / (4 pi perm0 sigma^3 epsilon)^1/2
- electric field = force/charge, where E* = E (4 pi perm0 sigma epsilon)^1/2 sigma / epsilon
- density = mass/volume, where rho* = rho sigma^dim

For style *real*, these are the units:

- mass = grams/mole
- distance = Angstroms
- time = femtoseconds
- energy = Kcal/mole
- velocity = Angstroms/femtosecond

- force = Kcal/mole−Angstrom
- torque = Kcal/mole
- temperature = degrees K
- pressure = atmospheres
- dynamic viscosity = Poise
- charge = multiple of electron charge (+1.0 is a proton)
- dipole = charge*Angstroms
- electric field = volts/Angstrom
- density = gram/cm^dim

For style *metal*, these are the units:

- mass = grams/mole
- distance = Angstroms
- time = picoseconds
- energy = eV
- velocity = Angstroms/picosecond
- force = eV/Angstrom
- torque = eV
- temperature = degrees K
- pressure = bars
- dynamic viscosity = Poise
- charge = multiple of electron charge (+1.0 is a proton)
- dipole = charge*Angstroms
- electric field = volts/Angstrom
- density = gram/cm^dim

For style *si*, these are the units:

- mass = kilograms
- distance = meters
- time = seconds
- energy = Joules
- velocity = meters/second
- force = Newtons
- torque = Newton−meters
- temperature = degrees K
- pressure = Pascals
- dynamic viscosity = Pascal*second
- charge = Coulombs
- dipole = Coulombs*meters
- electric field = volts/meter
- density = kilograms/meter^dim

For style *cgs*, these are the units:

- mass = grams
- distance = centimeters
- time = seconds
- energy = ergs
- velocity = centimeters/second
- force = dynes

- torque = dyne−centimeters
- temperature = degrees K
- pressure = dyne/cm^2 or barye = 1.0e−6 bars
- dynamic viscosity = Poise
- charge = statcoulombs or esu
- dipole = statcoul−cm = 10^18 debye
- electric field = statvolt/cm or dyne/esu
- density = grams/cm^dim

For style *electron*, these are the units:

- mass = atomic mass units
- distance = Bohr
- time = femtoseconds
- energy = Hartrees
- velocity = Bohr/atomic time units [1.03275e−15 seconds]
- force = Hartrees*Bohr
- temperature = degrees K
- pressure = Pascals
- charge = multiple of electron charge (+1.0 is a proton)
- dipole moment = Debye
- electric field = volts/cm

The units command also sets the timestep size and neighbor skin distance to default values for each style:

- For style *lj* these are dt = 0.005 tau and skin = 0.3 sigma.
- For style *real* these are dt = 1.0 fmsec and skin = 2.0 Angstroms.
- For style *metal* these are dt = 0.001 psec and skin = 2.0 Angstroms.
- For style *si* these are dt = 1.0e−8 sec and skin = 0.001 meters.
- For style *cgs* these are dt = 1.0e−8 sec and skin = 0.1 cm.
- For style *electron* these are dt = 0.001 fmsec and skin = 2.0 Bohr.

Restrictions:

This command cannot be used after the simulation box is defined by a read_data or create_box command.

Related commands: none

Default:

```
units lj
```

variable command

Syntax:

```
variable name style args ...
```

- name = name of variable to define
- style = *delete* or *index* or *loop* or *world* or *universe* or *uloop* or *string* or *equal* or *atom*

```
  delete = no args
    index args = one or more strings
    loop args = N
      N = integer size of loop, loop from 1 to N inclusive
    loop args = N pad
      N = integer size of loop, loop from 1 to N inclusive
      pad = all values will be same length, e.g. 001, 002, ..., 100
    loop args = N1 N2
      N1,N2 = loop from N1 to N2 inclusive
    loop args = N1 N2 pad
      N1,N2 = loop from N1 to N2 inclusive
      pad = all values will be same length, e.g. 050, 051, ..., 100
    world args = one string for each partition of processors
    universe args = one or more strings
    uloop args = N
      N = integer size of loop
    uloop args = N pad
      N = integer size of loop
      pad = all values will be same length, e.g. 001, 002, ..., 100
    string arg = one string
    equal or atom args = one formula containing numbers, thermo keywords, math operations, group
      numbers = 0.0, 100, -5.4, 2.8e-4, etc
      constants = PI
      thermo keywords = vol, ke, press, etc from thermo_style
      math operators = (), -x, x+y, x-y, x*y, x/y, x^y,
                       x==y, x!=y, xy, x>=y, xx||y, !x
      math functions = sqrt(x), exp(x), ln(x), log(x),
                       sin(x), cos(x), tan(x), asin(x), acos(x), atan(x), atan2(y,x),
                       random(x,y,z), normal(x,y,z), ceil(x), floor(x), round(x)
                       ramp(x,y), stagger(x,y), logfreq(x,y,z), vdisplace(x,y), swiggle(x,y,z),
      group functions = count(group), mass(group), charge(group),
                        xcm(group,dim), vcm(group,dim), fcm(group,dim),
                        bound(group,xmin), gyration(group), ke(group),
                        angmom(group,dim), torque(group,dim),
                        inertia(group,dimdim), omega(group,dim)
      region functions = count(group,region), mass(group,region), charge(group,region),
                         xcm(group,dim,region), vcm(group,dim,region), fcm(group,dim,region),
                         bound(group,xmin,region), gyration(group,region), ke(group,reigon),
                         angmom(group,dim,region), torque(group,dim,region),
                         inertia(group,dimdim,region), omega(group,dim,region)
      special functions = sum(x), min(x), max(x), ave(x), trap(x), gmask(x), rmask(x), grmask(x,
      atom value = mass[i], type[i], x[i], y[i], z[i], vx[i], vy[i], vz[i], fx[i], fy[i], fz[i]
      atom vector = mass, type, x, y, z, vx, vy, vz, fx, fy, fz
      compute references = c_ID, c_ID[i], c_ID[i][j]
      fix references = f_ID, f_ID[i], f_ID[i][j]
      variable references = v_name, v_name[i]
```

Examples:

```
variable x index run1 run2 run3 run4 run5 run6 run7 run8
```

```
variable LoopVar loop $n
variable beta equal temp/3.0
variable b1 equal x[234]+0.5*vol
variable b1 equal "x[234] + 0.5*vol"
variable b equal xcm(mol1,x)/2.0
variable b equal c_myTemp
variable b atom x*y/vol
variable foo string myfile
variable temp world 300.0 310.0 320.0 ${Tfinal}
variable x universe 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
variable x uloop 15 pad
variable x delete
```

Description:

This command assigns one or more strings to a variable name for evaluation later in the input script or during a simulation.

Variables can be used in several ways in LAMMPS. A variable can be referenced elsewhere in an input script to become part of a new input command. For variable styles that store multiple strings, the next command can be used to increment which string is assigned to the variable. Variables of style *equal* store a formula which when evaluated produces a single numeric value which can be output either directly (see the print, fix print, and run every commands) or as part of thermodynamic output (see the thermo_style command), or used as input to an averaging fix (see the fix ave/time command). Variables of style *atom* store a formula which when evaluated produces one numeric value per atom which can be output to a dump file (see the dump custom command) or used as input to an averaging fix (see the fix ave/spatial and fix ave/atom commands).

In the discussion that follows, the "name" of the variable is the arbitrary string that is the 1st argument in the variable command. This name can only contain alphanumeric characters and underscores. The "string" is one or more of the subsequent arguments. The "string" can be simple text as in the 1st example above, it can contain other variables as in the 2nd example, or it can be a formula as in the 3rd example. The "value" is the numeric quantity resulting from evaluation of the string. Note that the same string can generate different values when it is evaluated at different times during a simulation.

IMPORTANT NOTE: When the input script line that defines a variable of style *equal* or *atom* that contain a formula is encountered, the formula is NOT immediately evaluated and the result stored. See the discussion below about "Immediate Evaluation of Variables" if you want to do this.

IMPORTANT NOTE: When a variable command is encountered in the input script and the variable name has already been specified, the command is ignored. This means variables can NOT be re−defined in an input script (with 2 exceptions, read further). This is to allow an input script to be processed multiple times without resetting the variables; see the jump or include commands. It also means that using the command−line switch −var will override a corresponding index variable setting in the input script.

There are two exceptions to this rule. First, variables of style *string* and *equal* and *atom* ARE redefined each time the command is encountered. This allows these style of variables to be redefined multiple times in an input script. In a loop, this means the formula associated with an *equal* or *atom* style variable can change if it contains a substitution for another variable, e.g. $x.

Second, as described below, if a variable is iterated on to the end of its list of strings via the next command, it is removed from the list of active variables, and is thus available to be re−defined in a subsequent variable command. The *delete* style does the same thing.

This section of the manual explains how occurrences of a variable name in an input script line are replaced by the

variable's string. The variable name can be referenced as $x if the name "x" is a single character, or as ${LoopVar} if the name "LoopVar" is one or more characters.

As described below, for variable styles *index*, *loop*, *universe*, and *uloop*, which string is assigned to a variable can be incremented via the next command. When there are no more strings to assign, the variable is exhausted and a flag is set that causes the next jump command encountered in the input script to be skipped. This enables the construction of simple loops in the input script that are iterated over and then exited from.

As explained above, an exhausted variable can be re−used in an input script. The *delete* style also removes the variable, the same as if it were exhausted, allowing it to be redefined later in the input script or when the input script is looped over. This can be useful when breaking out of a loop via the if and jump commands before the variable would become exhausted. For example,

```
label       loop
variable    a loop 5
print       "A = $a"
if          "$a > 2" then "jump in.script break"
next        a
jump        in.script loop
label       break
variable    a delete
```

For the *index* style, one or more strings are specified. Initially, the 1st string is assigned to the variable. Each time a next command is used with the variable name, the next string is assigned. All processors assign the same string to the variable.

Index style variables with a single string value can also be set by using the command−line switch −var; see this section for details.

The *loop* style is identical to the *index* style except that the strings are the integers from 1 to N inclusive, if only one argument N is specified. This allows generation of a long list of runs (e.g. 1000) without having to list N strings in the input script. Initially, the string "1" is assigned to the variable. Each time a next command is used with the variable name, the next string ("2", "3", etc) is assigned. All processors assign the same string to the variable. The *loop* style can also be specified with two arguments N1 and N2. In this case the loop runs from N1 to N2 inclusive, and the string N1 is initially assigned to the variable.

For the *world* style, one or more strings are specified. There must be one string for each processor partition or "world". See this section of the manual for information on running LAMMPS with multiple partitions via the "−partition" command−line switch. This variable command assigns one string to each world. All processors in the world are assigned the same string. The next command cannot be used with *equal* style variables, since there is only one value per world. This style of variable is useful when you wish to run different simulations on different partitions, or when performing a parallel tempering simulation (see the temper command), to assign different temperatures to different partitions.

For the *universe* style, one or more strings are specified. There must be at least as many strings as there are processor partitions or "worlds". See this page for information on running LAMMPS with multiple partitions via the "−partition" command−line switch. This variable command initially assigns one string to each world. When a next command is encountered using this variable, the first processor partition to encounter it, is assigned the next available string. This continues until all the variable strings are consumed. Thus, this command can be used to run 50 simulations on 8 processor partitions. The simulations will be run one after the other on whatever partition becomes available, until they are all finished. *Universe* style variables are incremented using the files "tmp.lammps.variable" and "tmp.lammps.variable.lock" which you will see in your directory during such a LAMMPS run.

The *uloop* style is identical to the *universe* style except that the strings are the integers from 1 to N. This allows generation of long list of runs (e.g. 1000) without having to list N strings in the input script.

For the *equal* and *atom* styles, a single string is specified which represents a formula that will be evaluated afresh each time the variable is used. If you want spaces in the string, enclose it in double quotes so the parser will treat it as a single argument. For *equal* style variables the formula computes a scalar quantity, which becomes the value of the variable whenever it is evaluated. For *atom* style variables the formula computes one quantity for each atom whenever it is evaluated.

Note that *equal* and *atom* variables can produce different values at different stages of the input script or at different times during a run. For example, if an *equal* variable is used in a fix print command, different values could be printed each timestep it was invoked. If you want a variable to be evaluated immediately, so that the result is stored by the variable instead of the string, see the section below on "Immediate Evaluation of Variables".

The next command cannot be used with *equal* or *atom* style variables, since there is only one string.

The formula for an *equal* or *atom* variable can contain a variety of quantities. The syntax for each kind of quantity is simple, but multiple quantities can be nested and combined in various ways to build up formulas of arbitrary complexity. For example, this is a valid (though strange) variable formula:

```
variable x equal "pe + c_MyTemp / vol^(1/3)"
```

Specifically, an formula can contain numbers, thermo keywords, math operators, math functions, group functions, region functions, atom values, atom vectors, compute references, fix references, and references to other variables.

Number	0.2, 100, 1.0e20, −15.4, etc
Constant	PI
Thermo keywords	vol, pe, ebond, etc
Math operators	(), −x, x+y, x−y, x*y, x/y, x^y, x==y, x!=y, xy, x>=y, xx||y, !x
Math functions	sqrt(x), exp(x), ln(x), log(x), sin(x), cos(x), tan(x), asin(x), acos(x), atan(x), atan2(y,x), random(x,y,z), normal(x,y,z), ceil(x), floor(x), round(x), ramp(x,y), stagger(x,y), logfreq(x,y,z), vdisplace(x,y), swiggle(x,y,z), cwiggle(x,y,z)
Group functions	count(ID), mass(ID), charge(ID), xcm(ID,dim), vcm(ID,dim), fcm(ID,dim), bound(ID,dir), gyration(ID), ke(ID), angmom(ID,dim), torque(ID,dim), inertia(ID,dimdim), omega(ID,dim)
Region functions	count(ID,IDR), mass(ID,IDR), charge(ID,IDR), xcm(ID,dim,IDR), vcm(ID,dim,IDR), fcm(ID,dim,IDR), bound(ID,dir,IDR), gyration(ID,IDR), ke(ID,IDR), angmom(ID,dim,IDR), torque(ID,dim,IDR), inertia(ID,dimdim,IDR), omega(ID,dim,IDR)
Special functions	sum(x), min(x), max(x), ave(x), trap(x), gmask(x), rmask(x), grmask(x,y)
Atom values	mass[i], type[i], x[i], y[i], z[i], vx[i], vy[i], vz[i], fx[i], fy[i], fz[i]
Atom vectors	mass, type, x, y, z, vx, vy, vz, fx, fy, fz
Compute references	c_ID, c_ID[i], c_ID[i][j]
Fix references	f_ID, f_ID[i], f_ID[i][j]
Other variables	v_name, v_name[i]

Most of the formula elements produce a scalar value. A few produce a per−atom vector of values. These are the atom vectors, compute references that represent a per−atom vector, fix references that represent a per−atom vector, and variables that are atom−style variables. Math functions that operate on scalar values produce a scalar

value; math function that operate on per−atom vectors do so element−by−element and produce a per−atom vector.

A formula for equal−style variables cannot use any formula element that produces a per−atom vector. A formula for an atom−style variable can use formula elements that produce either a scalar value or a per−atom vector. Atom−style variables are evaluated by other commands that define a group on which they operate, e.g. a dump or compute or fix command. When they invoke the atom−style variable, only atoms in the group are inlcuded in the formula evaluation. The variable evaluates to 0.0 for atoms not in the group.

The thermo keywords allowed in a formula are those defined by the thermo_style custom command. Thermo keywords that require a compute to calculate their values such as "temp" or "press", use computes stored and invoked by the thermo_style command. This means that you can only use those keywords in a variable if the style you are using with the thermo_style command (and the thermo keywords associated with that style) also define and use the needed compute. Note that some thermo keywords use a compute indirectly to calculate their value (e.g. the enthalpy keyword uses temp, pe, and pressure). If a variable is evaluated directly in an input script (not during a run), then the values accessed by the thermo keyword must be current. See the discussion below about "Variable Accuracy".

Math Operators

Math operators are written in the usual way, where the "x" and "y" in the examples can themselves be arbitrarily complex formulas, as in the examples above. In this syntax, "x" and "y" can be scalar values or per−atom vectors. For example, "ke/natoms" is the division of two scalars, where "vy+vz" is the element−by−element sum of two per−atom vectors of y and z velocities.

Operators are evaluated left to right and have the usual C−style precedence: unary minus and unary logical NOT operator "!" have the highest precedence, exponentiation "^" is next; multiplication and division are next; addition and subtraction are next; the 4 relational operators "", and ">=" are next; the two remaining relational operators "==" and "!=" are next; then the logical AND operator " and finally the logical OR operator "||" has the lowest precedence. Parenthesis can be used to group one or more portions of a formula and/or enforce a different order of evaluation than what would occur with the default precedence.

The 6 relational operators return either a 1.0 or 0.0 depending on whether the relationship between x and y is TRUE or FALSE. For example the expression x

These relational and logical operators can be used as a masking or selection operation in a formula. For example, the number of atoms whose properties satifsy one or more criteria could be calculated by taking the returned per−atom vector of ones and zeroes and passing it to the compute reduce command.

Math Functions

Math functions are specified as keywords followed by one or more parenthesized arguments "x", "y", "z", each of which can themselves be arbitrarily complex formulas. In this syntax, the arguments can represent scalar values or per−atom vectors. In the latter case, the math operation is performed on each element of the vector. For example, "sqrt(natoms)" is the sqrt() of a scalar, where "sqrt(y*z)" yields a per−atom vector with each element being the sqrt() of the product of one atom's y and z coordinates.

Most of the math functions perform obvious operations. The ln() is the natural log; log() is the base 10 log.

The random(x,y,z) function takes 3 arguments: x = lo, y = hi, and z = seed. It generates a uniform random number between lo and hi. The normal(x,y,z) function also takes 3 arguments: x = mu, y = sigma, and z = seed. It generates a Gaussian variate centered on mu with variance sigma^2. In both cases the seed is used the first time the internal random number generator is invoked, to initialize it. For equal−style variables, every processor uses

the same seed so that they each generate the same sequence of random numbers. For atom−style variables, a unique seed is created for each processor, based on the specified seed. This effectively generates a different random number for each atom being looped over in the atom−style variable.

IMPORTANT NOTE: Internally, there is just one random number generator for all equal−style variables and one for all atom−style variables. If you define multiple variables (of each style) which use the random() or normal() math functions, then the internal random number generators will only be initialized once, which means only one of the specified seeds will determine the sequence of generated random numbers.

The ceil(), floor(), and round() functions are those in the C math library. Ceil() is the smallest integer not less than its argument. Floor() if the largest integer not greater than its argument. Round() is the nearest integer to its argument.

The ramp(x,y) function uses the current timestep to generate a value linearly intepolated between the specified x,y values over the course of a run, according to this formula:

```
value = x + (y−x) * (timestep−startstep) / (stopstep−startstep)
```

The run begins on startstep and ends on stopstep. Startstep and stopstep can span multiple runs, using the *start* and *stop* keywords of the run command. See the run command for details of how to do this.

The stagger(x,y) function uses the current timestep to generate a new timestep. X,y > 0 and x > y is required. The generated timesteps increase in a staggered fashion, as the sequence x,x+y,2x,2x+y,3x,3x+y,etc. For any current timestep, the next timestep in the sequence is returned. Thus if stagger(1000,100) is used in a variable by the dump_modify every command, it will generate the sequence of output timesteps:

```
100,1000,1100,2000,2100,3000,etc
```

The logfreq(x,y,z) function uses the current timestep to generate a new timestep. X,y,z > 0 and y < z is required. The generated timesteps increase in a logarithmic fashion, as the sequence x,2x,3x,...y*x,z*x,2*z*x,3*z*x,...y*z*x,z*z*x,2*z*x*x,etc. For any current timestep, the next timestep in the sequence is returned. Thus if logfreq(100,4,10) is used in a variable by the dump_modify every command, it will generate the sequence of output timesteps:

```
100,200,300,400,1000,2000,3000,4000,10000,20000,etc
```

The vdisplace(x,y) function takes 2 arguments: x = coord0 and y = velocity, and uses the elapsed time to change the coordinate value by a linear displacement due to the applied velocity over the course of a run, according to this formula:

```
value = coord0 + velocity*(timestep−startstep)*dt
```

where dt = the timestep size.

The run begins on startstep. Startstep can span multiple runs, using the *start* keyword of the run command. See the run command for details of how to do this. Note that the thermo_style keyword elaplong = timestep−startstep.

The swiggle(x,y,z) and cwiggle(x,y,z) functions each take 3 arguments: x = coord0, y = amplitude, z = period. They use the elapsed time to oscillate the coordinate value by a sin() or cos() function over the course of a run, according to one of these formulas, where omega = 2 PI / period:

```
value = coord0 + Amplitude * sin(omega*(timestep−startstep)*dt)
value = coord0 + Amplitude * (1 − cos(omega*(timestep−startstep)*dt))
```

where dt = the timestep size.

The run begins on startstep. Startstep can span multiple runs, using the *start* keyword of the run command. See the run command for details of how to do this. Note that the thermo_style keyword elaplong = timestep−startstep.

Group and Region Functions

Group functions are specified as keywords followed by one or two parenthesized arguments. The first argument is the group−ID. The *dim* argument, if it exists, is *x* or *y* or *z*. The *dir* argument, if it exists, is *xmin*, *xmax*, *ymin*, *ymax*, *zmin*, or *zmax*. The *dimdim* argument, if it exists, is *xx* or *yy* or *zz* or *xy* or *yz* or *xz*.

The group function count() is the number of atoms in the group. The group functions mass() and charge() are the total mass and charge of the group. Xcm() and vcm() return components of the position and velocity of the center of mass of the group. Fcm() returns a component of the total force on the group of atoms. Bound() returns the min/max of a particular coordinate for all atoms in the group. Gyration() computes the radius−of−gyration of the group of atoms. See the compute gyration command for a definition of the formula. Angmom() returns components of the angular momentum of the group of atoms around its center of mass. Torque() returns components of the torque on the group of atoms around its center of mass, based on current forces on the atoms. Inertia() returns one of 6 components of the inertia tensor of the group of atoms around its center of mass. Omega() returns components of the angular velocity of the group of atoms around its center of mass.

Region functions are specified exactly the same way as group functions except they take an extra argument which is the region ID. The function is computed for all atoms that are in both the group and the region. If the group is "all", then the only criteria for atom inclusion is that it be in the region.

Special Functions

Special functions take specific kinds of arguments, meaning their arguments cannot be formulas themselves.

The sum(x), min(x), max(x), ave(x), and trap(x) functions each take 1 argument which is of the form "c_ID" or "c_ID[N]" or "f_ID" or "f_ID[N]". The first two are computes and the second two are fixes; the ID in the reference should be replaced by the ID of a compute or fix defined elsewhere in the input script. The compute or fix must produce either a global vector or array. If it produces a global vector, then the notation without "[N]" should be used. If it produces a global array, then the notation with "[N]" should be used, when N is an integer, to specify which column of the global array is being referenced.

These functions operate on the global vector of inputs and reduce it to a single scalar value. This is analagous to the operation of the compute reduce command, which invokes the same functions on per−atom and local vectors.

The sum() function calculates the sum of all the vector elements. The min() and max() functions find the minimum and maximum element respectively. The ave() function is the same as sum() except that it divides the result by the length of the vector. The trap() function is the same as sum() except the first and last elements are multiplied by a weighting factor of 1/2 when performing the sum. This effectively implements an integratiion via the trapezoidal rule on the global vector of data. I.e. consider a set of points, equally spaced by 1 in their x coordinate: (1,V1), (2,V2), ..., (N,VN), where the Vi are the values in the global vector of length N. The integral from 1 to N of these points is trap(). When appropriately normalized by the timestep size, this function is useful for calculating integrals of time−series data, like that generated by the fix ave/correlate command.

The gmask(x) function takes 1 argument which is a group ID. It can only be used in atom−style variables. It returns a 1 for atoms that are in the group, and a 0 for atoms that are not.

The rmask(x) function takes 1 argument which is a region ID. It can only be used in atom−style variables. It returns a 1 for atoms that are in the geometric region, and a 0 for atoms that are not.

The grmask(x,y) function takes 2 arguments. The first is a group ID, and the second is a region ID. It can only be used in atom−style variables. It returns a 1 for atoms that are in both the group and region, and a 0 for atoms that are not in both.

Atom Values and Vectors

Atom values take a single integer argument I from 1 to N, where I is the an atom−ID, e.g. x[243], which means use the x coordinate of the atom with ID = 243.

Atom vectors generate one value per atom, so that a reference like "vx" means the x−component of each atom's velocity will be used when evaluating the variable. Note that other atom attributes can be used as inputs to a variable by using the compute property/atom command and then specifying a quantity from that compute.

Compute References

Compute references access quantities calculated by a compute. The ID in the reference should be replaced by the ID of a compute defined elsewhere in the input script. As discussed in the doc page for the compute command, computes can produce global, per−atom, or local values. Only global and per−atom values can be used in a variable. Computes can also produce a scalar, vector, or array. An equal−style variable can only use scalar values, which means a global scalar, or an element of a global or per−atom vector or array. Atom−style variables can use the same scalar values. They can also use per−atom vector values. A vector value can be a per−atom vector itself, or a column of an per−atom array. See the doc pages for individual computes to see what kind of values they produce.

Examples of different kinds of compute references are as follows. There is no ambiguity as to what a reference means, since computes only produce global or per−atom quantities, never both.

c_ID	global scalar, or per−atom vector
c_ID[I]	Ith element of global vector, or atom I's value in per−atom vector, or Ith column from per−atom array
c_ID[I][J]	I,J element of global array, or atom I's Jth value in per−atom array

If a variable containing a compute is evaluated directly in an input script (not during a run), then the values accessed by the compute must be current. See the discussion below about "Variable Accuracy".

Fix References

Fix references access quantities calculated by a fix. The ID in the reference should be replaced by the ID of a fix defined elsewhere in the input script. As discussed in the doc page for the fix command, fixes can produce global, per−atom, or local values. Only global and per−atom values can be used in a variable. Fixes can also produce a scalar, vector, or array. An equal−style variable can only use scalar values, which means a global scalar, or an element of a global or per−atom vector or array. Atom−style variables can use the same scalar values. They can also use per−atom vector values. A vector value can be a per−atom vector itself, or a column of an per−atom array. See the doc pages for individual fixes to see what kind of values they produce.

The different kinds of fix references are exactly the same as the compute references listed in the above table, where "c_" is replaced by "f_". Again, there is no ambiguity as to what a reference means, since fixes only produce global or per−atom quantities, never both.

f_ID	global scalar, or per−atom vector
f_ID[I]	Ith element of global vector, or atom I's value in per−atom vector, or Ith column from per−atom array
f_ID[I][J]	I,J element of global array, or atom I's Jth value in per−atom array

If a variable containing a fix is evaluated directly in an input script (not during a run), then the values accessed by the fix should be current. See the discussion below about "Variable Accuracy".

Note that some fixes only generate quantities on certain timesteps. If a variable attempts to access the fix on non−allowed timesteps, an error is generated. For example, the fix ave/time command may only generate averaged quantities every 100 steps. See the doc pages for individual fix commands for details.

Variable References

Variable references access quantities calulated by other variables, which will cause those variables to be evaluated. The name in the reference should be replaced by the name of a variable defined elsewhere in the input script. As discussed on this doc page, atom−style variables generate a per−atom vector of values; all other variable styles generate a global scalar value. An equal−style variable can reference a global scalar value produced by another variable, but not a per−atom vector produced by an atom−style variable. Atom−style variables can reference either global scalar or per−atom vector values produced by kind of variable.

Examples of different kinds of variable references are as follows. There is no ambiguity as to what a reference means, since variables produce only a global scalar or a per−atom vectors, never both.

v_name	scalar, or per−atom vector
v_name[I]	atom I's value in per−atom vector

IMPORTANT NOTE: If you define variables in circular manner like this:

```
variable a equal v_b
variable b equal v_a
print $a
```

then LAMMPS may run for a while when the print statement is invoked!

Immediate Evaluation of Variables:

There is a difference between referencing a variable with a leading $ sign (e.g. $x or ${abc}) versus with a leading "v_" (e.g. v_x or v_abc). The former can be used in any command, including a variable command, to force the immediate evaluation of the referenced variable and the substitution of its value into the command. The latter is a required kind of argument to some commands (e.g. the fix ave/spatial or dump custom or thermo_style commands) if you wish it to evaluate a variable periodically during a run. It can also be used in a variable formula if you wish to reference a second variable. The second variable will be evaluated whenever the first variable is evaluated.

As an example, suppose you use this command in your input script to define the variable "v" as

```
variable v equal vol
```

before a run where the simulation box size changes. You might think this will assign the initial volume to the variable "v". That is not the case. Rather it assigns a formula which evaluates the volume (using the thermo_style keyword "vol") to the variable "v". If you use the variable "v" in some other command like fix ave/time then the current volume of the box will be evaluated continuously during the run.

If you want to store the initial volume of the system, you can do it this way:

```
variable v equal vol
variable v0 equal $v
```

The second command will force "v" to be evaluated (yielding the initial volume) and assign that value to the variable "v0". Thus the command

```
thermo_style custom step v_v v_v0
```

would print out both the current and initial volume periodically during the run.

Note that it is a mistake to enclose a variable formula in double quotes if it contains variables preceeded by $ signs. For example,

```
variable vratio equal "${vfinal}/${v0}"
```

This is because the quotes prevent variable substitution (see this section on parsing input script commands), and thus an error will occur when the formula for "vratio" is evaluated later.

Variable Accuracy:

Obviously, LAMMPS attempts to evaluate variables containing formulas (*equal* and *atom* style variables) accurately whenever the evaluation is performed. Depending on what is included in the formula, this may require invoking a compute, either directly or indirectly via a thermo keyword, or accessing a value previously calculated by a compute, or accessing a value calculated and stored by a fix. If the compute is one that calculates the pressure or energy of the system, then these quantities need to be tallied during the evaluation of the interatomic potentials (pair, bond, etc) on timesteps that the variable will need the values.

LAMMPS keeps track of all of this during a run or energy minimization. An error will be generated if you attempt to evaluate a variable on timesteps when it cannot produce accurate values. For example, if a thermo_style custom command prints a variable which accesses values stored by a fix ave/time command and the timesteps on which thermo output is generated are not multiples of the averaging frequency used in the fix command, then an error will occur.

An input script can also request variables be evaluated before or after or in between runs, e.g. by including them in a print command. In this case, if a compute is needed to evaluate a variable (either directly or indirectly), LAMMPS will not invoke the compute, but it will use a value previously calculated by the compute, and can do this only if it is current. Fixes will always provide a quantity needed by a variable, but the quantity may or may not be current. This leads to one of three kinds of behavior:

(1) The variable may be evaluated accurately. If it contains references to a compute or fix, and these values were calculated on the last timestep of a preceeding run, then they will be accessed and used by the variable and the result will be accurate.

(2) LAMMPS may not be able to evaluate the variable and generate an error. For example, if the variable requires a quantity from a compute that is not current, LAMMPS will generate an error. This means, for example, that such a variable cannot be evaluated before the first run has occurred. Likewise, in between runs, such a variable cannot be accessed unless it was evaluated on the last timestep of the preceding run, e.g. by thermodynamic output.

One way to get around this problem is to perform a 0−timestep run before using the variable. For example, these commands

```
variable t equal temp
print "Initial temperature = $t"
run 1000
```

will generate an error if the run is the first run specified in the input script, because generating a value for the "t" variable requires a compute for calculating the temperature to be invoked.

However, this sequence of commands would be fine:

```
run 0
variable t equal temp
print "Initial temperature = $t"
run 1000
```

The 0−timestep run initializes and invokes various computes, including the one for temperature, so that the value it stores is current and can be accessed by the variable "t" after the run has completed. Note that a 0−timestep run does not alter the state of the system, so it does not change the input state for the 1000−timestep run that follows. Also note that the 0−timestep run must actually use and invoke the compute in question (e.g. via thermo or dump output) in order for it to enable the compute to be used in a variable after the run. Thus if you are trying to print a variable that uses a compute you have defined, you could insure it was invoked on the last timestep of the preceding run by including it in thermodynamic output.

Unlike computes, fixes will never generate an error if their values are accessed by a variable in between runs. They always return some value to the variable. However, the value may not be what you expect if the fix has not yet calculated the quantity of interest or it is not current. For example, the fix indent command stores the force on the indenter. But this is not computed until a run is performed. Thus if a variable attempts to print this value before the first run, zeroes will be output. Again, performing a 0−timestep run before printing the variable has the desired effect.

(3) The variable may be evaluated incorrectly. And LAMMPS may have no way to detect this has occurred. Consider the following sequence of commands:

```
pair_coeff 1 1 1.0 1.0
run 1000
pair_coeff 1 1 1.5 1.0
variable e equal pe
print "Final potential energy = $e"
```

The first run is performed using one setting for the pairwise potential defined by the pair_style and pair_coeff commands. The potential energy is evaluated on the final timestep and stored by the compute pe compute (this is done by the thermo_style command). Then a pair coefficient is changed, altering the potential energy of the system. When the potential energy is printed via the "e" variable, LAMMPS will use the potential energy value stored by the compute pe compute, thinking it is current. There are many other commands which could alter the state of the system between runs, causing a variable to evaluate incorrectly.

The solution to this issue is the same as for case (2) above, namely perform a 0−timestep run before the variable is evaluated to insure the system is up−to−date. For example, this sequence of commands would print a potential energy that reflected the changed pairwise coefficient:

```
pair_coeff 1 1 1.0 1.0
run 1000
pair_coeff 1 1 1.5 1.0
run 0
variable e equal pe
print "Final potential energy = $e"
```

Restrictions:

Indexing any formula element by global atom ID, such as an atom value, requires the atom style to use a global mapping in order to look up the vector indices. By default, only atom styles with molecular information create global maps. The atom_modify map command can override the default.

All *universe*− and *uloop*−style variables defined in an input script must have the same number of values.

Related commands:

next, jump, include, temper, fix print, print

Default: none

velocity command

Syntax:

```
velocity group-ID style args keyword value ...
```

- group−ID = ID of group of atoms whose velocity will be changed
- style = *create* or *set* or *scale* or *ramp* or *zero*

```
  create args = temp seed
      temp = temperature value (temperature units)
      seed = random # seed (positive integer)
    set args = vx vy vz
      vx,vy,vz = velocity value or NULL (velocity units)
      any of vx,vy,vz van be a variable (see below)
    scale arg = temp
      temp = temperature value (temperature units)
    ramp args = vdim vlo vhi dim clo chi
      vdim = vx or vy or vz
      vlo,vhi = lower and upper velocity value (velocity units)
      dim = x or y or z
      clo,chi = lower and upper coordinate bound (distance units)
    zero arg = linear or angular
      linear = zero the linear momentum
      angular = zero the angular momentum
```

- zero or more keyword/value pairs may be appended
- keyword = *dist* or *sum* or *mom* or *rot* or *temp* or *loop* or *units*

```
  dist value = uniform or gaussian
    sum value = no or yes
    mom value = no or yes
    rot value = no or yes
    temp value = temperature ID
    loop value = all or local or geom
    units value = box or lattice
```

Examples:

```
velocity all create 300.0 4928459 rot yes dist gaussian
velocity border set NULL 4.0 v_vz sum yes units box
velocity flow scale 300.0
velocity flow ramp vx 0.0 5.0 y 5 25 temp mytemp
velocity all zero linear
```

Description:

Set or change the velocities of a group of atoms in one of several styles. For each style, there are required arguments and optional keyword/value parameters. Not all options are used by each style. Each option has a default as listed below.

The *create* style generates an ensemble of velocities using a random number generator with the specified seed as the specified temperature.

The *set* style sets the velocities of all atoms in the group to the specified values. If any component is specified as NULL, then it is not set. Any of the vx,vy,vz velocity components can be specified as an equal−style or

atom−style variable. If the value is a variable, it should be specified as v_name, where name is the variable name. In this case, the variable will be evaluated, and its value used to determine the velocity component.

Equal−style variables can specify formulas with various mathematical functions, and include thermo_style command keywords for the simulation box parameters or other parameters.

Atom−style variables can specify the same formulas as equal−style variables but can also include per−atom values, such as atom coordinates. Thus it is easy to specify a spatially−dependent velocity field.

The *scale* style computes the current temperature of the group of atoms and then rescales the velocities to the specified temperature.

The *ramp* style is similar to that used by the compute temp/ramp command. Velocities ramped uniformly from vlo to vhi are applied to dimension vx, or vy, or vz. The value assigned to a particular atom depends on its relative coordinate value (in dim) from clo to chi. For the example above, an atom with y−coordinate of 10 (1/4 of the way from 5 to 25), would be assigned a x−velocity of 1.25 (1/4 of the way from 0.0 to 5.0). Atoms outside the coordinate bounds (less than 5 or greater than 25 in this case), are assigned velocities equal to vlo or vhi (0.0 or 5.0 in this case).

The *zero* style adjusts the velocities of the group of atoms so that the aggregate linear or angular momentum is zero. No other changes are made to the velocities of the atoms.

All temperatures specified in the velocity command are in temperature units; see the units command. The units of velocities and coordinates depend on whether the *units* keyword is set to *box* or *lattice*, as discussed below.

For all styles, no atoms are assigned z−component velocities if the simulation is 2d; see the dimension command.

The keyword/value option pairs are used in the following ways by the various styles.

The *dist* option is used by *create*. The ensemble of generated velocities can be a *uniform* distribution from some minimum to maximum value, scaled to produce the requested temperature. Or it can be a *gaussian* distribution with a mean of 0.0 and a sigma scaled to produce the requested temperature.

The *sum* option is used by all styles, except *zero*. The new velocities will be added to the existing ones if sum = yes, or will replace them if sum = no.

The *mom* and *rot* options are used by *create*. If mom = yes, the linear momentum of the newly created ensemble of velocities is zeroed; if rot = yes, the angular momentum is zeroed.

The *temp* option is used by *create* and *scale* to specify a compute that calculates temperature in a desired way. If this option is not specified, *create* and *scale* calculate temperature using a compute that is defined as follows:

```
compute velocity_temp group−ID temp
```

where group−ID is the same ID used in the velocity command. i.e. the group of atoms whose velocity is being altered. This compute is deleted when the velocity command is finished. See the compute temp command for details. If the computed temperature should have degrees−of−freedom removed due to fix constraints (e.g. SHAKE or rigid−body constraints), then the appropriate fix command must be specified before the velocity command is issued.

The *loop* option is used by *create* in the following ways.

If loop = all, then each processor loops over all atoms in the simulation to create velocities, but only stores velocities for atoms it owns. This can be a slow loop for a large simulation. If atoms were read from a data file, the velocity assigned to a particular atom will be the same, independent of how many processors are being used. This will not be the case if atoms were created using the create_atoms command, since atom IDs will likely be assigned to atoms differently.

If loop = local, then each processor loops over only its atoms to produce velocities. The random number seed is adjusted to give a different set of velocities on each processor. This is a fast loop, but the velocity assigned to a particular atom will depend on which processor owns it. Thus the results will always be different when a simulation is run on a different number of processors.

If loop = geom, then each processor loops over only its atoms. For each atom a unique random number seed is created, based on the atom's xyz coordinates. A velocity is generated using that seed. This is a fast loop and the velocity assigned to a particular atom will be the same, independent of how many processors are used. However, the set of generated velocities may be more correlated than if the *all* or *local* options are used.

Note that the *loop geom* option will not necessarily assign identical velocities for two simulations run on different machines. This is because the computations based on xyz coordinates are sensitive to tiny differences in the double−precision value for a coordinate as stored on a particular machine.

The *units* option is used by *set* and *ramp*. If units = box, the velocities and coordinates specified in the velocity command are in the standard units described by the units command (e.g. Angstroms/fmsec for real units). If units = lattice, velocities are in units of lattice spacings per time (e.g. spacings/fmsec) and coordinates are in lattice spacings. The lattice command must have been previously used to define the lattice spacing.

Restrictions: none

Related commands:

fix shake, lattice

Default:

The option defaults are dist = uniform, sum = no, mom = yes, rot = no, temp = full style on group−ID, loop = all, and units = lattice.

write_restart command

Syntax:

```
write_restart file
```

- file = name of file to write restart information to

Examples:

```
write_restart restart.equil
write_restart poly.%.*
```

Description:

Write a binary restart file of the current state of the simulation. See the read_restart command for information about what is stored in a restart file.

During a long simulation, the restart command is typically used to dump restart files periodically. The write_restart command is useful after a minimization or whenever you wish to write out a single current restart file.

Similar to dump files, the restart filename can contain two wild−card characters. If a "*" appears in the filename, it is replaced with the current timestep value. If a "%" character appears in the filename, then one file is written by each processor and the "%" character is replaced with the processor ID from 0 to P−1. An additional file with the "%" replaced by "base" is also written, which contains global information. For example, the files written for filename restart.% would be restart.base, restart.0, restart.1, ... restart.P−1. This creates smaller files and can be a fast mode of output and subsequent input on parallel machines that support parallel I/O.

Restart files can be read by a read_restart command to restart a simulation from a particular state. Because the file is binary (to enable exact restarts), it may not be readable on another machine. In this case, the restart2data program in the tools directory can be used to convert a restart file to an ASCII data file. Both the read_restart command and restart2data tool can read in a restart file that was written with the "%" character so that multiple files were created.

Restrictions:

This command requires inter−processor communication to migrate atoms before the restart file is written. This means that your system must be ready to perform a simulation before using this command (force fields setup, atom masses initialized, etc).

Related commands:

restart, read_restart

Default: none