

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
1. Introduction.....	3
1.1 What is LAMMPS.....	3
1.2 LAMMPS features.....	4
Kinds of systems LAMMPS can simulate:.....	4
Force fields:.....	4
Creation of atoms:.....	4
Ensembles, constraints, and boundary conditions:.....	5
Integrators:.....	5
Output:.....	5
Pre- and post-processing:.....	5
1.3 LAMMPS non-features.....	6
1.4 Open source distribution.....	7
1.5 Acknowledgments and citations.....	8
2. Getting Started.....	11
2.1 What's in the LAMMPS distribution.....	11
2.2 Making LAMMPS.....	11
2.3 Making LAMMPS with optional packages.....	14
2.4 Building LAMMPS as a library.....	16
2.5 Running LAMMPS.....	17
2.6 Command-line options.....	17
2.7 LAMMPS screen output.....	19
2.8 Tips for users of previous LAMMPS versions.....	20
3. Commands.....	21
3.1 LAMMPS input script.....	21
3.2 Parsing rules.....	22
3.3 Input script structure.....	22
3.4 Commands listed by category.....	23
3.5 Individual commands.....	24
4. How-to discussions.....	27
4.1 Restarting a simulation.....	27
4.2 2d simulations.....	28
4.3 CHARMM and AMBER force fields.....	29
4.4 Running multiple simulations from one input script.....	29
4.5 Parallel tempering.....	31
4.6 Granular models.....	31
4.7 TIP3P water model.....	32
4.8 TIP4P water model.....	33
4.9 SPC water model.....	34
4.10 Coupling LAMMPS to other codes.....	34
4.11 Visualizing LAMMPS snapshots.....	36
4.12 Non-orthogonal simulation boxes.....	36
4.13 NEMD simulations.....	37
4.14 Extended spherical and aspherical particles.....	38
4.15 Output from LAMMPS.....	40
4.16 Thermostatting, barostatting, and computing temperature.....	43
5. Example problems.....	46
6. Performance & scalability.....	48

Table of Contents

7. Additional tools.....	49
amber2lmp tool.....	49
binary2txt tool.....	50
ch2lmp tool.....	50
chain tool.....	50
data2xmovie tool.....	50
eam generate tool.....	50
lmp2arc tool.....	51
lmp2cfg tool.....	51
lmp2traj tool.....	51
lmp2vmd tool.....	51
matlab tool.....	51
micelle2d tool.....	52
msi2lmp tool.....	52
pymol_asphere tool.....	52
python tool.....	52
restart2data tool.....	52
thermo_extract tool.....	53
vim tool.....	53
xmovie tool.....	53
8. Modifying & extending LAMMPS.....	54
Atom styles.....	56
Bond, angle, dihedral, improper potentials.....	56
Compute styles.....	57
Dump styles.....	57
Dump custom output options.....	57
Fix styles.....	58
Input script commands.....	59
Kspace computations.....	59
Minimization solvers.....	60
Pairwise potentials.....	60
Region styles.....	60
Thermodynamic output options.....	61
Variable options.....	61
Submitting new features to the developers to include in LAMMPS.....	62
9. Errors.....	63
9.1 Common problems.....	63
9.2 Reporting bugs.....	64
9.3 Error & warning messages.....	64
Errors:.....	64
Warnings:.....	98
10. Future and history.....	102
10.1 Coming attractions.....	102
10.2 Past versions.....	102
angle_style charmm command.....	104
angle_style class2 command.....	105
angle_style cg/cmm command.....	107
angle_coeff command.....	108

Table of Contents

angle_style cosine command.....	110
angle_style cosine/delta command.....	111
angle_style cosine/squared command.....	112
angle_style harmonic command.....	113
angle_style hybrid command.....	114
angle_style none command.....	115
angle_style command.....	116
atom_modify command.....	118
atom_style command.....	119
bond_style class2 command.....	121
bond_coeff command.....	122
bond_style fene command.....	124
bond_style fene/expand command.....	125
bond_style harmonic command.....	127
bond_style hybrid command.....	128
bond_style morse command.....	129
bond_style none command.....	130
bond_style nonlinear command.....	131
bond_style quartic command.....	132
bond_style command.....	134
boundary command.....	136
change_box command.....	137
clear command.....	138
communicate command.....	139
compute command.....	141
compute ackland/atom command.....	144
compute centro/atom command.....	146
compute coord/atom command.....	147
compute damage/atom command.....	148
compute displace/atom command.....	149
compute erotate/asphere command.....	150
compute erotate/sphere command.....	151
compute group/group command.....	152
compute ke command.....	153
compute ke/atom command.....	154
compute_modify command.....	155
compute pe command.....	156
compute pe/atom command.....	157
compute pressure command.....	159
compute reduce command.....	161
compute stress/atom command.....	163
compute temp command.....	165
compute temp/asphere command.....	167
compute temp/com command.....	169
compute temp/deform command.....	171
compute temp/partial command.....	173
compute temp/ramp command.....	175
compute temp/region command.....	177

Table of Contents

compute temp/sphere command.....	179
create_atoms command.....	181
create_box command.....	183
delete_atoms command.....	184
delete_bonds command.....	186
dielectric command.....	188
dihedral_style charmm command.....	189
dihedral_style class2 command.....	191
dihedral_coeff command.....	194
dihedral_style harmonic command.....	196
dihedral_style helix command.....	197
dihedral_style hybrid command.....	198
dihedral_style multi/harmonic command.....	199
dihedral_style none command.....	200
dihedral_style opls command.....	201
dihedral_style command.....	202
dimension command.....	204
dipole command.....	205
displace_atoms command.....	206
displace_box command.....	208
dump command.....	211
dump_modify command.....	216
echo command.....	218
fix command.....	219
fix addforce command.....	222
fix ave/atom command.....	223
fix ave/spatial command.....	225
fix ave/time command.....	229
fix aveforce command.....	232
fix bond/break command.....	233
fix bond/create command.....	236
fix bond/swap command.....	239
fix com command.....	242
fix coord/original command.....	243
fix deform command.....	244
fix deposit command.....	249
fix drag command.....	252
fix dt/reset command.....	253
fix efield command.....	255
fix enforce2d command.....	256
fix freeze command.....	257
fix gravity command.....	258
fix gyration command.....	260
fix heat command.....	261
fix indent command.....	263
fix langevin command.....	266
fix lineforce command.....	269
fix_modify command.....	270

Table of Contents

fix momentum command.....	271
fix msd command.....	273
fix nph command.....	275
fix npt command.....	278
fix npt/asphere command.....	282
fix npt/sphere command.....	286
fix nve command.....	290
fix nve/asphere command.....	291
fix nve/limit command.....	292
fix nve/noforce command.....	294
fix nve/sphere command.....	295
fix nvt command.....	296
fix nvt/asphere command.....	299
fix nvt/sllod command.....	302
fix nvt/sphere command.....	305
fix orient/fcc command.....	308
fix planeforce command.....	312
fix poems.....	313
fix pour command.....	315
fix press/berendsen command.....	317
fix print command.....	320
fix rdf command.....	322
fix recenter command.....	324
fix rigid.....	326
fix setforce command.....	328
fix shake command.....	329
fix smd command.....	331
fix spring command.....	333
fix spring/rg command.....	335
fix spring/self command.....	337
fix temp/berendsen command.....	338
fix temp/rescale command.....	340
fix thermal/conductivity command.....	342
fix tmd command.....	344
fix viscosity command.....	346
fix viscous command.....	349
fix wall/gran command.....	351
fix wall/lj126 command.....	353
fix wall/lj93 command.....	355
fix wall/reflect command.....	357
fix wiggle command.....	359
group command.....	360
if command.....	362
improper_style class2 command.....	364
improper_coeff command.....	366
improper_style cvff command.....	368
improper_style harmonic command.....	369
improper_style hybrid command.....	370

Table of Contents

improper_style none command.....	371
improper_style command.....	372
include command.....	374
jump command.....	375
kspace_modify command.....	377
kspace_style command.....	379
label command.....	381
lattice command.....	382
log command.....	385
mass command.....	386
min_modify command.....	387
min_style command.....	388
minimize command.....	389
neigh_modify command.....	393
neighbor command.....	396
newton command.....	398
next command.....	399
orient command.....	401
origin command.....	402
pair_style airebo command.....	403
pair_style buck command.....	406
pair_style buck/coul/cut command.....	406
pair_style buck/coul/long command.....	406
pair_style buck/coul command.....	408
pair_style lj/charmm/coul/charmm command.....	411
pair_style lj/charmm/coul/charmm/implicit command.....	411
pair_style lj/charmm/coul/long command.....	411
pair_style lj/charmm/coul/long/opt command.....	411
pair_style lj/class2 command.....	414
pair_style lj/class2/coul/cut command.....	414
pair_style lj/class2/coul/long command.....	414
pair_style cg/cmm command.....	417
pair_style cg/cmm/coul/cut command.....	417
pair_style cg/cmm/coul/long command.....	417
pair_coeff command.....	420
pair_style colloid command.....	423
pair_style coul/cut command.....	426
pair_style coul/debye command.....	426
pair_style coul/long command.....	426
pair_style dipole/cut command.....	428
pair_style dpd command.....	431
pair_style eam command.....	433
pair_style eam/opt command.....	433
pair_style eam/alloy command.....	433
pair_style eam/alloy/opt command.....	433
pair_style eam/fs command.....	433
pair_style eam/fs/opt command.....	433
pair_style gayberne command.....	439

Table of Contents

pair_style gran/hertzian command.....	442
pair_style gran/history command.....	442
pair_style gran/no_history command.....	442
pair_style lj/gromacs command.....	444
pair_style lj/gromacs/coul/gromacs command.....	444
pair_style hybrid command.....	446
pair_style hybrid/overlay command.....	446
pair_style lj/cut command.....	449
pair_style lj/cut/opt command.....	449
pair_style lj/cut/coul/cut command.....	449
pair_style lj/cut/coul/debye command.....	449
pair_style lj/cut/coul/long command.....	449
pair_style lj/cut/coul/long/tip4p command.....	449
pair_style lj/coul command.....	453
pair_style lj/expand command.....	456
pair_style lj/smooth command.....	458
pair_style lubricate command.....	460
pair_style meam command.....	462
pair_modify command.....	466
pair_style morse command.....	469
pair_style morse/opt command.....	469
pair_style none command.....	471
pair_style peri/pmb command.....	472
pair_style resquared command.....	474
pair_style soft command.....	477
pair_style command.....	479
pair_style sw command.....	482
pair_style table command.....	485
pair_style tersoff command.....	488
pair_style tersoff/zbl command.....	492
pair_write command.....	497
pair_style yukawa command.....	499
print command.....	501
processors command.....	502
read_data command.....	503
read_restart command.....	513
region command.....	515
replicate command.....	517
reset_timestep command.....	518
restart command.....	519
run command.....	521
run_style command.....	524
set command.....	527
shape command.....	530
shell command.....	532
special_bonds command.....	534
temper command.....	536
thermo command.....	538

Table of Contents

thermo_modify command.....	539
thermo_style command.....	541
timestep command.....	545
uncompute command.....	546
undump command.....	547
unfix command.....	548
units command.....	549
variable command.....	552
velocity command.....	560
write_restart command.....	563

LAMMPS Documentation

(6 Jan 2009 version 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 developers of LAMMPS are [Steve Plimpton](#), Paul Crozier, and Aidan Thompson who can be contacted at sjplimp,pscrozi,athomps@sandia.gov. The [LAMMPS WWW Site](http://lammps.sandia.gov) 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](http://lammps.sandia.gov/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. [How-to discussions](#)
 - 4.1 [Restarting a simulation](#)
 - 4.2 [2d simulations](#)
 - 4.3 [CHARMM and AMBER force fields](#)
 - 4.4 [Running multiple simulations from one input script](#)
 - 4.5 [Parallel tempering](#)
 - 4.6 [Granular models](#)
 - 4.7 [TIP3P water model](#)

- 4.8 [TIP4P water model](#)
- 4.9 [SPC water model](#)
- 4.10 [Coupling LAMMPS to other codes](#)
- 4.11 [Visualizing LAMMPS snapshots](#)
- 4.12 [Non-orthogonal simulation boxes](#)
- 4.13 [NEMD simulations](#)
- 4.14 [Extended spherical and aspherical particles](#)
- 4.15 [Output from LAMMPS](#)
- 4.16 [Thermostatting, barostatting, and compute temperature](#)
- 5. [Example problems](#)
- 6. [Performance & scalability](#)
- 7. [Additional tools](#)
- 8. [Modifying & Extending LAMMPS](#)
- 9. [Errors](#)
 - 9.1 [Common problems](#)
 - 9.2 [Reporting bugs](#)
 - 9.3 [Error & warning messages](#)
- 10. [Future and history](#)
 - 10.1 [Coming attractions](#)
 - 10.2 [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.

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.

Kinds of systems LAMMPS can simulate:

([atom style](#) command)

- atomic (e.g. box of Lennard–Jonesium)
- bead–spring polymers
- united–atom polymers or organic molecules
- all–atom polymers, organic molecules, proteins, DNA
- metals
- granular materials
- coarse–grained mesoscale models
- ellipsoidal particles
- point dipolar particles
- hybrid systems

Force fields:

([pair style](#), [bond style](#), [angle style](#), [dihedral style](#), [improper style](#), [kspace style](#) commands)

- pairwise potentials: Lennard–Jones, Buckingham, Morse, Yukawa, soft, class 2 (COMPASS), tabulated
- charged pairwise potentials: Coulombic, point–dipole
- manybody potentials: EAM, Finnis/Sinclair EAM, modified EAM (MEAM), Stillinger–Weber, Tersoff, AI–REBO
- coarse–grain potentials: granular, DPD, GayBerne, REsquared, colloidal
- bond potentials: harmonic, FENE, Morse, nonlinear, class 2, quartic (breakable)
- angle potentials: harmonic, CHARMM, cosine, cosine/squared, class 2 (COMPASS)
- dihedral potentials: harmonic, CHARMM, multi–harmonic, helix, class 2 (COMPASS), OPLS
- improper potentials: harmonic, cvff, class 2 (COMPASS)
- hybrid potentials: multiple pair, bond, angle, dihedral, improper potentials can be used in one simulation
- overlaid potentials: superposition of multiple pair potentials
- 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
- CHARMM, AMBER, OPLS, GROMACS, force–field compatibility

Creation of atoms:

([read_data](#), [lattice](#), [create_atoms](#), [delete_atoms](#), [displace_atoms](#) 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)
- 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 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
- independent or coupled rigid body integration
- SHAKE bond and angle constraints
- walls of various kinds
- targeted molecular dynamics (TMD) and steered molecule dynamics (SMD) constraints
- non-equilibrium molecular dynamics (NEMD)
- variety of additional boundary conditions and constraints

Integrators:

([run](#), [run_style](#), [temper](#) commands)

- velocity-Verlet integrator
- Brownian dynamics
- energy minimization via conjugate gradient relaxation
- rRESPA hierarchical timestepping
- parallel tempering (replica exchange)
- run multiple independent simulations simultaneously

Output:

([dump](#), [restart](#) commands)

- log file of thermodynamic info
- text dump files of atom coords, velocities, other per-atom quantities
- binary restart files
- per-atom quantities (energy, stress, centro-symmetry parameter, 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 formats

Pre- and post-processing:

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](#).

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 automatically
- 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](#), [athomp](#) 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.

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)
AI–REBO 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 unpack, make, 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
doc	documentation
examples	simple test problems
potentials	embedded atom method (EAM) potential files
src	source files
tools	pre- and post-processing tools

2.2 Making LAMMPS

Read this first:

Building LAMMPS can be non-trivial. You will likely need to edit a makefile, there are compiler options, additional libraries can be used (MPI, FFT), 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 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 (which there isn't a similar Makefile for in the

distribution), send it to the developers and we'll include it in future LAMMPS releases.

Building a LAMMPS executable:

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 several 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
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 typically 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.

Errors that can occur when making LAMMPS:

(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 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 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.

(2) Other errors typically occur because the low-level Makefile isn't setup correctly for your machine. If your platform is named "foo", you need to create a Makefile.foo in the MAKE sub-directory. Use whatever existing file is closest to your platform as a starting point. See the next section for more instructions.

Editing a new low-level Makefile.foo:

These are the issues you need to address when editing a low-level Makefile for your machine. With a couple exceptions, the only portion of the file you should need to edit is the "System-specific Settings" section.

(1) Change the first line of Makefile.foo to include the word "foo" and whatever other options you set. This is the line you will see if you just type "make".

(2) Set the paths and flags for your C++ compiler, including optimization flags. You can use g++, the open-source GNU compiler, which is available on all Unix systems. Vendor compilers often produce faster code. On boxes with Intel CPUs, we suggest using the free Intel icc compiler, which you can download from [Intel's compiler site](#).

(3) If you want LAMMPS to run in parallel, you must have an MPI library installed on your platform. If you do not use "mpicc" as your compiler/linker, then Makefile.foo needs to specify where the mpi.h file (-I switch) and the libmpi.a library (-L switch) is found. If you are installing MPI yourself, we recommend Argonne's MPICH 1.2 which can be downloaded from the [Argonne MPI 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 LAM, so find out how to build and link with it. If you use MPICH 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 may arise when linking LAMMPS to the MPI library.

(4) If you just want LAMMPS to run on a single processor, you can use the STUBS library in place of MPI, since you don't need an MPI library installed on your system. See the Makefile.serial file for how to specify the `-I` and `-L` switches. You will also need to build the STUBS library for your platform before making LAMMPS itself. From the STUBS dir, type "make" and it will hopefully create a libmpi.a suitable for linking to LAMMPS. If the build fails, you will need to edit the STUBS/Makefile for your platform.

The file STUBS/mpi.cpp has a CPU timer function `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.

(5) If you want to use the particle-particle particle-mesh (PPPM) option in LAMMPS for long-range Coulombics, you must have a 1d FFT library installed on your platform. This is specified by a switch of the form `-DFFT_XXX` where XXX = INTEL, DEC, SGI, SCSL, or FFTW. All but the last one are native vendor-provided libraries. FFTW is a fast, portable library that should work on any platform. You can download it from www.fftw.org. Use version 2.1.X, not the newer 3.0.X. Building FFTW for your box should be as simple as `./configure; make`. Whichever FFT library you have on your platform, you'll need to set the appropriate `-I` and `-L` switches in Makefile.foo.

If you examine `fft3d.c` and `fft3d.h` you'll see it's possible to add other vendor FFT libraries via `#ifdef` statements in the appropriate places. If you successfully add a new FFT option, like `-DFFT_IBM`, please send the [developers](#) an email; we'd like to add it to LAMMPS.

(6) If you don't plan to use PPPM, you don't need an FFT library. Use a `-DFFT_NONE` switch in the CCFLAGS setting of Makefile.foo, or exclude the KSPACE package (see below).

(7) There are a few other `-D` compiler switches you can set as part of CCFLAGS. 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. 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. If you compile with `-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.

(8) The DEPFLAGS setting is how the C++ compiler creates a dependency file for each 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 (a long list of errors involving *.d files), then you'll need to create a Makefile.foo patterned after Makefile.tflow, which uses different rules that do not involve dependency files.

That's it. Once you have a correct Makefile.foo and you have pre-built the MPI and FFT libraries it will use, all you need to do from the src directory is type one of these 2 commands:

```
make foo
gmake foo
```

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

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" will delete all *.o object files created when LAMMPS is built.

(3) On some machines with some compiler options, the Coulomb tabling option that is enabled by default for "long" pair styles such as `lj/cut/coul/long` and `lj/charmm/coul/long` does not work. Tables are used by these styles since it can offer a 2x speed-up. A symptom of this problem is getting wildly large energies on timestep 0 of the examples/peptide simulation.

Here are several work-arounds. Coulomb tables can be disabled by setting "table 0" in the `pair_modify` command.

The associated files (e.g. `pair_lj_cut_coul_long.cpp`) can be compiled at a lower optimization level like `-O2`, or with the compiler flag `-fno-strict-aliasing`. The latter can be done by adding something like these lines in your `Makefile.machine`:

```
NOALIAS =          -fno-strict-aliasing

pair_lj_cut_coul_long.o : pair_lj_cut_coul_long.cpp
    $(CC) $(CFLAGS) $(NOALIAS) -c $<

pair_lj_charmm_coul_long.o : pair_lj_charmm_coul_long.cpp
    $(CC) $(CFLAGS) $(NOALIAS) -c $<
```

On a Macintosh, try compiling the pair "long" files without the `-fast` compiler option.

(4) Building for a Macintosh.

OS X is BSD Unix, so it already works. See the `Makefile.mac` file.

(5) Building for MicroSoft Windows.

I've never done this, but LAMMPS is just standard C++ with MPI and FFT calls. You can use cygwin to build LAMMPS with a Unix make; see `Makefile.cygwin`. Or you should be able to pull all the source files into Visual C++ (ugh) or some similar development environment and build it. In the `src/MAKE/Windows` directory are some notes from users on how they built LAMMPS under Windows, so you can look at their instructions for tips. Good luck – we can't help you on this one.

2.3 Making LAMMPS with optional packages

The source code for LAMMPS is structured as a large set of core files which are always used, plus optional packages, which 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 both standard and user-contributed 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
dpd	dissipative particle dynamics (DPD) force field
granular	force fields and boundary conditions for granular systems
kspc	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
poems	coupled rigid body motion
xtc	dump atom snapshots in XTC format

There are also user-contributed packages which may be as simple as a single additional file 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.

Any or all packages can be included or excluded when LAMMPS is built. The one exception is that to use the standard "opt" package, you must also be using the "molecule" and "manybody" packages. You may wish to exclude certain packages if you will never run certain kinds of simulations. This will keep you from having to build auxiliary libraries (see below) and will produce a smaller executable which may run a bit faster.

By default, LAMMPS includes only the "kspc", "manybody", and "molecule" packages. As described below, some standard packages require LAMMPS be linked to separately built library files, which will require editing of your src/MAKE/Makefile.machine.

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. These commands work by simply moving files back and forth between the main src directory and sub-directories with the package name, 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 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 directories if the package has been included. It should be used after a patch is installed, since patches only update the master package version of a file. Typing "make package-overwrite" will overwrite files in the package directories with src files. Typing "make package-check" will list differences between src and package versions of the same files.

To use the "meam" package you must build LAMMPS with the MEAM library in lib/meam, which computes the modified embedded atom method potential, which is a generalization of EAM potentials that can be used to model a wider variety of materials. This MEAM implementation was written by Greg Wagner at Sandia. To build

LAMMPS with MEAM, you must use a low-level LAMMPS Makefile that includes the MEAM directory in its paths. See `Makefile.linux_meam` as an example. You must also build MEAM itself as a library before building LAMMPS, so that LAMMPS can link against it. This requires a F90 compiler. The library is built by typing "make" from within the meam directory with the appropriate Makefile, e.g. "make -f Makefile.icc". If one of the provided Makefiles is not appropriate for your system you can edit or add one as needed.

Note that linking a Fortran library to a C++ code can be problematic (e.g. Fortran routine names can't be found due to non-standard underscore rules) and typically requires additional C++ or F90 libraries be included in the link. You may need to read documentation for your compiler about how to do this correctly.

To use the "poems" package you must build LAMMPS with the POEMS library in `lib/poems`, which computes the constrained rigid-body motion of articulated (jointed) multibody systems. POEMS was written and is distributed by Prof Kurt Anderson's group at Rensselaer Polytechnic Institute (RPI). To build LAMMPS with POEMS, you must use a low-level LAMMPS Makefile that includes the POEMS directory in its paths. See `Makefile.g++_poems` as an example. You must also build POEMS itself as a library before building LAMMPS, so that LAMMPS can link against it. The POEMS library is built by typing "make" from within the poems directory with the appropriate Makefile, e.g. "make -f Makefile.g++". If one of the provided Makefiles is not appropriate for your system you can edit or add one as needed.

2.4 Building LAMMPS as a library

LAMMPS 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. 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. See the sample code `examples/couple/c++_driver.cpp` as an example.

When used from a C or Fortran program or a scripting language, the library has a simple function-style interface, provided in `library.cpp` and `library.h`. See the sample code `examples/couple/c_driver.cpp` as an example.

You can add as many functions as you wish to `library.cpp` and `library.h`. In a general sense, those functions can access LAMMPS data and return it to the caller or set LAMMPS data values as specified by the caller. These 4 functions are currently included in `library.cpp`:

```
void lammps_open(int, char **, MPI_Comm, void **ptr);
void lammps_close(void *ptr);
int lammps_file(void *ptr, char *);
int lammps_command(void *ptr, 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 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 should be used in subsequent library calls. Note that `lammps_open()` can be called multiple times to create multiple LAMMPS objects.

The `lammmps_close()` function is used to shut down LAMMPS and free all its memory. The `lammmps_file()` and `lammmps_command()` functions are used to pass a file or string to LAMMPS as if it were an input file or single command read from an input script.

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 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 one of the Lennard–Jones tests on a Linux box, using `mpirun` to launch a parallel job:

```
cd src
make linux
cp lmp_linux ../examples/lj
cd ../examples/lj
mpirun -np 4 lmp_linux <in.lj.nve
```

The screen output from LAMMPS is described in the next section. As it runs, LAMMPS also writes a `log.lammmps` 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. For example, `lmp_ibm` might be launched as follows:

```
mpirun -np 16 lmp_ibm -var f tmp.out -log my.log -screen none <in.alloy
```

These are the command-line options:

```
-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.

`-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.

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.

`-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.

`-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.

`-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.

`-var name value`

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}`). The value can be any string. Using this command-line option is equivalent to putting the line "variable name index value" 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 $5N\log_2(N)$, 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

LAMMPS 2003 is a complete C++ 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 LAMMPS 2003.

If you are a previous user of LAMMPS 2001, these are the most significant changes you will notice in LAMMPS 2003:

- (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 LAMMPS 2003, 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 LAMMPS 2003.
- (4) Binary restart files written by LAMMPS 2001 cannot be read by LAMMPS 2003 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 LAMMPS 2003 [read_data](#) command to read it in.
- (5) There are numerous small numerical changes in LAMMPS 2003 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 an 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 line ends with a `"` character (with no trailing whitespace), 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.
- (3) The line is searched repeatedly for `$` characters which indicate variables that are replaced with a text string. If the `$` is followed by curly brackets, then the variable name is the text inside the curly brackets. If no curly brackets follow the `$`, then the variable name is the character immediately following the `$`. 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 scripts.
- (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) Text with spaces can be enclosed in double quotes so it will be treated as a single argument. See the [dump](#) [modify](#) or [fix print](#) commands for examples. A `#` or `$` character that is in text between double quotes will not be treated as a comment or substituted for as a variable.

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](#), [dipole](#), [group](#), [mass](#), [min_modify](#), [min_style](#), [neigh_modify](#), [neighbor](#), [reset_timestep](#), [run_style](#), [set](#), [shape](#), [timestep](#), [velocity](#)

Fixes:

[fix](#), [fix_modify](#), [unfix](#)

Computes:

[compute](#), [compute_modify](#), [uncompute](#)

Output:

[dump](#), [dump_modify](#), [restart](#), [thermo](#), [thermo_modify](#), [thermo_style](#), [undump](#), [write_restart](#)

Actions:

[delete_atoms](#), [delete_bonds](#), [displace_atoms](#), [displace_box](#), [minimize](#), [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	dipole	displace_atoms	displace_box	dump
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	neigh_modify	neighbor	newton	next	pair_coeff
pair_modify	pair_style	pair_write	print	processors	read_data
read_restart	region	replicate	reset_timestep	restart	run

run_style	set	shape	shell	special_bonds	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:

addforce	aveforce	ave/atom	ave/spatial	ave/time	bond/break	bond/create	bond/swap
com	coord/original	deform	deposit	drag	dt/reset	efield	enforce2d
freeze	gravity	gyration	heat	indent	langevin	lineforce	msd
momentum	nph	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	rdf	recenter	rigid	setforce
shake	spring	spring/rg	spring/self	temp/berendsen	temp/rescale	thermal/conductivity	tmd
viscosity	viscous	wall/gran	wall/lj126	wall/lj93	wall/reflect	wiggle	

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

[smd](#)

Compute styles. See the [compute](#) command for one-line descriptions of each style or click on the style itself for a full description:

centro/atom	coord/atom	damage/atom	displace/atom	erotate/asphere	erotate/sphere
group/group	ke	ke/atom	pe	pe/atom	pressure
reduce	stress/atom	temp	temp/asphere	temp/com	temp/deform
temp/partial	temp/ramp	temp/region	temp/sphere		

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

[ackland/atom](#)

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	airebo
buck	buck/coul/cut	buck/coul/long	colloid
coul/cut	coul/debye	coul/long	dipole/cut
dpd	eam	eam/opt	eam/alloy
eam/alloy/opt	eam/fs	eam/fs/opt	gayberne
gran/hertzian	gran/history	gran/no_history	lj/charmm/coul/charmm
lj/charmm/coul/charmm/implicit	lj/charmm/coul/long	lj/charmm/coul/long/opt	lj/class2
lj/class2/coul/cut	lj/class2/coul/long	lj/cut	lj/cut/opt
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
lubricate	meam	morse	morse/opt

resquared	peri/pmb	soft	sw
table	tersoff	tersoff/zbl	yukawa

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

buck/coul	cg/cmm	cg/cmm/coul/cut	cg/cmm/coul/long
lj/coul			

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			

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/squared	harmonic

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

cg/cmm

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

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			

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/tip4p
-----------------------	----------------------	----------------------------

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

ewald/n

4. How-to discussions

The following sections describe what commands can be used to perform certain kinds of LAMMPS simulations.

- [4.1 Restarting a simulation](#)
- [4.2 2d simulations](#)
- [4.3 CHARMM and AMBER force fields](#)
- [4.4 Running multiple simulations from one input script](#)
- [4.5 Parallel tempering](#)
- [4.6 Granular models](#)
- [4.7 TIP3P water model](#)
- [4.8 TIP4P water model](#)
- [4.9 SPC water model](#)
- [4.10 Coupling LAMMPS to other codes](#)
- [4.11 Visualizing LAMMPS snapshots](#)
- [4.12 Non-orthogonal simulation boxes](#)
- [4.13 NEMD simulations](#)
- [4.14 Extended spherical and aspherical particles](#)
- [4.15 Output from LAMMPS](#)
- [4.16 Thermostatting, barostatting and computing temperature](#)

The example input scripts included in the LAMMPS distribution and highlighted in [this section](#) also show how to setup and run various kinds of problems.

4.1 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.

4.2 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.

4.3 CHARMM and AMBER force fields

There are many different ways to compute forces in the [CHARMM](#) and [AMBER](#) molecular dynamics codes, only some of which are available as options in LAMMPS. 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. 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
-

4.4 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.

4.5 Parallel tempering

The [temper](#) command can be used to perform a parallel tempering or replica–exchange simulation where multiple copies of a simulation are run at different temperatures on different sets of processors, and Monte Carlo temperature swaps are performed between pairs of copies.

Use the `–procs` and `–in` [command–line switches](#) to launch LAMMPS on multiple partitions.

In your input script, define a set of temperatures, one for each processor partition, using the [variable](#) command:

```
variable t world 300.0 310.0 320.0 330.0
```

Define a fix of style [nvt](#) or [langevin](#) to control the temperature of each simulation:

```
fix myfix all nvt $t $t 100.0
```

Use the [temper](#) command in place of a [run](#) command to perform a simulation where tempering exchanges will take place:

```
temper 100000 100 $t myfix 3847 58382
```

4.6 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](#) granular
- [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
-

4.7 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):

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

4.8 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

4.9 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 with long-range Coulombics (Ewald or PPPM in LAMMPS).

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

To use SPC with a long-range Coulombic solver (Ewald or PPPM in LAMMPS), the only parameters that change are the partial charge assignments:

O charge = –0.8476
H charge = 0.4238

4.10 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.

[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, or Fortran (or any other language that supports a vanilla C-like interface, e.g. a scripting language). 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. `Library.cpp` and `library.h` contain such a C interface with the functions:

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

The functions contain C++ code you could write in a C++ application that was invoking LAMMPS directly. Note that LAMMPS classes are defined within a LAMMPS namespace (`LAMMPS_NS`) if you use them from another C++ application.

Two of the routines in `library.cpp` are of particular note. The `lammps_open()` function initiates LAMMPS and takes an MPI communicator as an argument. It returns a pointer to a LAMMPS "object". As with C++, 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.

`Library.cpp` contains a `lammps_command()` function to which the caller passes a single LAMMPS command (a string). Thus the calling code can read or generate a series of LAMMPS commands (e.g. an input script) one line at a time and pass it thru the library interface to setup a problem and then run it.

A few other sample functions are included in `library.cpp`, but the key idea is that you can write any functions you wish to define an interface for how your code talks to LAMMPS and add them to `library.cpp` and `library.h`. The routines you add can access any LAMMPS data. The `examples/couple` directory has example C++ and C codes which show how a stand-alone code can link LAMMPS as a library, run LAMMPS on a subset of processors, grab data from LAMMPS, change it, and put it back into LAMMPS.

4.11 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 [ch2imp](#) tool, which contains a lammps2pdb Perl script which converts LAMMPS dump files into PDB files. The second is the [imp2arc](#) tool which converts LAMMPS dump files into Accelrys' Insight MD program files. The third is the [imp2cfg](#) 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.

4.12 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). If the box size is xprd by yprd by zprd then the 3 mutually orthogonal edge vectors of an orthogonal simulation box are $a = (xprd, 0, 0)$, $b = (0, yprd, 0)$, and $c = (0, 0, zprd)$.

LAMMPS also allows non-orthogonal simulation boxes (triclinic symmetry) to be defined with 3 additional "tilt" parameters which change the edge vectors of the simulation box to be $a = (xprd, 0, 0)$, $b = (xy, yprd, 0)$, and $c = (xz, yz, zprd)$. The xy, xz, and yz parameters can be positive or negative. The simulation box must be periodic in both dimensions associated with a tilt factor. For example, if $xz \neq 0.0$, then the x and z dimensions must be periodic.

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 all

equivalent.

You tell LAMMPS to use a non-orthogonal box when the simulation box is defined. This happens in one of 3 ways. If the [create_box](#) command is used with a region of style *prism*, then a non-orthogonal domain 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 non-orthogonal domain 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 non-orthogonal box will be enabled for the restarted simulation.

Note that you can define a non-orthogonal box with all 3 tilt factors = 0.0, so that it is initially orthogonal. This is necessary if the box will become non-orthogonal. Alternatively, you can use the [change_box](#) command to convert a simulation box from orthogonal to non-orthogonal and vice versa.

One use of non-orthogonal 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 non-orthogonal simulation box via the [create_atoms](#) command. Note that while the box edge vectors a,b,c cannot be arbitrary vectors (e.g. a must be aligned with the x axis), it is possible to rotate any crystal's basis vectors so that they meet these restrictions.

A second use of non-orthogonal boxes 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, and yz tilt factors as a simulation runs.

Another use of non-orthogonal boxes is to perform non-equilibrium MD (NEMD) simulations, as discussed in the next section.

4.13 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.

4.14 Extended spherical and aspherical particles

Typical MD models treat atoms or particles as point masses. Sometimes, however, it is desirable to have a model where the particles are extended spherioids or extended aspherical particles such as an ellipsoid. 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 3 [atom styles](#) that define extended particles: granular, dipole, ellipsoid.

Granular particles are extended spherioids and each particle can have a unique diameter and mass (or density). These particles store an angular velocity (ω) and can be acted upon by torque.

Dipole particles are extended spherioids with a point dipole and each particle type has a diameter and mass, set by the [shape](#) and [mass](#) commands. These particles store an angular velocity (ω) and can be acted upon by torque. They also store an orientation for the point dipole (μ) which has a length set by the [dipole](#) command. The [set](#) command can be used to initialize the orientation of dipole moments.

Ellipsoid particles are aspherical. Each particle type has an ellipsoidal shape and mass, defined by the [shape](#) and [mass](#) commands. These particles store an angular momentum and their orientation (quaternion), and can be acted upon by torque. They do not store an angular velocity (ω) which can be in a different direction than angular momentum. The [set](#) command can be used to initialize the orientation of ellipsoidal particles and has a brief explanation of quaternions.

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 extended or aspherical. For example, if the 3 shape parameters are set to the same value, the particle will be a spheroid rather than an ellipsoid. If 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 (ellipsoid versus spheroid 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](#), extended spherioids and ellipsoids are still treated as 3d particles, rather than as disks or ellipses. This means they still have a moment of inertia for a 3d extended object. When their temperature is computed, 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 [atom_style granular](#). The [dipole pair style](#) is used with [atom_style dipole](#). The [GayBerne](#) and [REsquared](#) potentials require particles have a [shape](#) and are designed for [ellipsoidal particles](#). The [lubrication potential](#) requires that particles have a [shape](#). It can currently only be used with extended 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 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 thermostating. 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:

- [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.

(NOTE: 6/08 the feature described in the following paragraph has not yet been released. It will be soon.)

If any of the constituent particles of a rigid body are extended particles (spheroids 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.

4.15 Output from LAMMPS

Aside from [restart files](#), there are two basic kinds of LAMMPS output. The first is [thermodynamic output](#), which is a list of quantities printed every few timesteps to the screen and logfile. The second is [dump files](#), which contain snapshots of atoms and various per-atom values and are written at a specified frequency. A simulation prints one set of thermodynamic output; it may generate any number of dump files. As discussed below, LAMMPS gives you a variety of ways to determine what quantities are computed and printed when thermodynamic info or dump files are output. There are also three fixes which can do their own output of user-chosen quantities: [fix ave/time](#) for time averaging, [fix ave/spatial](#) for spatial averaging, and [fix print](#). These are also described below. Throughout this discussion, note that users can [add their own computes and fixes to LAMMPS](#) which can then generate values that can be output with these commands.

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. Each of these are described in turn.

In LAMMPS, a [compute](#) comes in two flavors: ones that compute global values (e.g. temperature, pressure tensor) and ones that compute per-atom values. Only global quantities from a compute can be used for thermodynamic output. The user-defined ID of the compute is used along with an optional subscript as part of the [thermo_style](#) command. E.g. c_myTemp outputs the single scalar value generated by the compute; c_myTemp[2] outputs the 2nd vector value. Note that there is a [compute reduce](#) command which can sum per-atom quantities into a global scalar or vector.

[Fixes](#) can generate global scalar or vector values which can be output with thermodynamic output, e.g. the energy of an indenter's interaction with the simulation atoms. These values are accessed via the same format as a compute's values, as `f_ID` or `f_ID[N]`. See the doc pages for individual fix commands to see which ones generate global values that can be output with thermodynamic info. The [fix ave/time](#) command generates time-averaged global quantities which can be accessed for thermodynamic output, e.g. a time-averaged pressure.

[Variables](#) can be defined in the input script. All styles except the atom-style variable can be used for thermodynamic output, since it generates per-atom values. A variable with the name "abc" is referenced in a `thermo_style` command as `v_abc`.

The formula associated with equal-style variables can contain math operations and functions (`x+y`, `x*y`, `()`, `sqrt`, `exp`, etc), atom values (`x[N]`, `fx[N]`, etc), group functions (`mass()`, `vcm()`, etc), references to thermodynamic quantities (`temp`, `press`, `vol`, etc), references to [computes](#) that generate global values, references to [fixes](#) that generate global values, or references to other [variables](#). Thus an equal-style variable is the most general way to define some quantity to calculate and include with thermodynamic output.

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 keywords can be specified (`tag`, `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 value to be output. Each of these are described in turn.

[Computes](#) that generate one or more per-atom values can be accessed by the `dump custom` command. These are computes that have the word "atom" in their style name, e.g. `ke/atom`, `stress/atom`, etc. The values are accessed as `c_myKE` for a scalar per-atom quantity or as `c_myStress[2]` for a component of a vector per-atom quantity.

[Fixes](#) can generate per-atom values to output to dump files. For example, the [fix ave/atom](#) command calculates time-averages of per-atom quantities, such as per-atom [computes](#) and atom-style [variables](#). These per-atom fix values are accessed by the [dump custom](#) command via the same format as a compute's values, as `f_myKE` or `f_myStress[2]`.

[Variables](#) can be defined in the input script. Only atom-style variables can be used for `dump custom` output, since only they produce per-atom values. A variable with the name "abc" is referenced in a `dump custom` command as `v_abc`.

Just like equal-style variables, the formula associated with atom-style variables can contain math operations and functions (`x+y`, `x*y`, `()`, `sqrt`, `exp`, etc), atom values (`x[N]`, `fx[N]`, etc), group functions (`mass()`, `vcm()`, etc), references to thermodynamic quantities (`temp`, `press`, `vol`, etc), references to [computes](#) that generate global values, references to [fixes](#) that generate global values, or references to non atom-style variables that generate global values. In addition, an atom-style variable can reference vectors of atom values (`x[]`, `fx[]`, etc), [computes](#) that generate per-atom values, [fixes](#) that generate per-atom values, or other atom-style variables. Thus an atom-style variable is the most general way to define some quantity to calculate and output to a dump file.

Fix output

Three other fixes are of particular note for output: [fix ave/time](#), [fix ave/spatial](#), and [fix print](#).

The [fix ave/time](#) command enables time-averaging of any global quantity, like those output with thermodynamic info. The user specifies one or more quantities to time average. 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`), any thermodynamic quantity can be

time averaged in this way.

The [fix ave/time](#) command offers several options for how it performs time averaging. The results it produces can be used in two ways. First, they can be written directly to a file, one line per timestamp. Note that the averaging parameters can be specified in such a way that averaging is not done at all, in which case this is simply a convenient means of outputting desired quantities directly to a separate file. Like other fixes that produce global quantities, the results of this fix can also be used as input to any command that accesses global quantities, e.g. by the [thermo_style custom](#) command, by a variable, etc.

The [fix ave/spatial](#) command enables spatial-averaging of 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 [fix ave/spatial](#) command offers several options for how it performs time averaging. The per-layer values it produces can be used in two ways. First, they can be written directly to a file. Note that the averaging parameters can be specified in such a way that time averaging is not done, in which case this is a convenient means of simply outputting desired quantities (summed over atoms within a 1d layer) directly to a separate file. Like other fixes that produce global quantities, the results of this fix can also be used as input by any command that accesses global quantities, e.g. by the [thermo_style custom](#) command, by a variable, etc. See the doc page for [fix ave/spatial](#) for a description of how these values are indexed.

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), and as explained above, variables themselves can contain references to global values generated by [thermodynamic keywords](#), [computes](#), [fixes](#), or other [variables](#). Thus the [fix print](#) command is a means to output any desired calculated quantity separate from normal thermodynamic or dump file output.

This table summarizes the various output options, specifying what their inputs and outputs are. The frequency with which they are invoked and produce output is also listed. Basically, any two commands can be hooked together so long as one produces output that matches the input needs of the other. A "match" means that the frequencies and global vs per-atom attributes are the same.

Command	Input	Input Freq	Output	Output Freq
thermo_style custom	thermo keyword, global scalar/vector compute, global scalar/vector fix, equal variable	nthermo	screen, log	nthermo
dump custom	keyword, peratom scalar/vector compute, peratom scalar/vector fix, atom variable	dump freq	file	dump freq
global fixes	N/A	N/A	global scalar/vector	1 or nevery
peratom fixes	N/A	N/A	peratom scalar/vector	1 or nevery
fix ave/time	global scalar/vector fix, global scalar/vector compute, equal variable	nevery	global scalar/vector, file	nfreq
fix ave/spatial	peratom scalar/vector fix, peratom scalar/vector compute, atom vector, atom variable, density mass/number	nevery	global vector	nfreq, file
fix ave/atom	peratom scalar/vector compute, peratom scalar/vector fix, atom variable, atom vector	nevery	peratom scalar/vector	nfreq
fix print	any variable	nevery	screen, file	nevery

global computes	N/A	N/A	global scalar/vector	N/A
peratom computes	N/A	N/A	peratom scalar/vector	N/A
compute sum	peratom scalar/vector compute, peratom scalar/vector fix, atom variable	N/A	global scalar/vector	N/A
variable equal	thermo keywords, atom value vx[123], global scalar/vector compute, global scalar/vector fix, non-atom variable	N/A	global scalar	N/A
variable atom	thermo keywords, atom value vx[123], atom vector vx[], global scalar/vector compute, peratom scalar/vector compute, global scalar/vector fix, peratom scalar/vector fix, any variable	N/A	peratom scalar	N/A
print	any variable	N/A	screen, log	between runs
run every	any variable	never	screen, log	never

4.16 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 motion 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/ramp](#)
- [compute temp/region](#)

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

Thermostatting in LAMMPS is performed by [fixes](#). Four thermostatting fixes are currently available: Nose–Hoover (nvt), Berendsen, Langevin, and direct rescaling (temp/rescale):

- [fix nvt](#)
- [fix nvt/sphere](#)
- [fix nvt/asphere](#)
- [fix nvt/sllod](#)
- [fix temp/berendsen](#)
- [fix langevin](#)
- [fix temp/rescale](#)

[Fix nvt](#) only thermostats the translational velocity of particles. [Fix nvt/sllod](#) does as well, 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.

Any of these 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 thermostating 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. 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 barostatting 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 thermostating. Both [fix nph](#) and [fix press/berendsen](#) can be used in conjunction with any of the thermostating 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. The barostatting fixes can also use temperature computes that remove bias for the purpose of computing the current temperature 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 thermostating or barostating 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.

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

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

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

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

5. 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
crack	crack propagation in a 2d solid
dipole	point dipolar particles, 2d system
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
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)
pour	pouring of granular particles into a 3d box, then chute flow
rigid	rigid bodies modeled as independent or coupled
shear	sideways shear applied to 2d solid, with and without a void

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 a directory "couple" in the examples sub-directory, which contains a stand-alone code umbrella.cpp that links LAMMPS as a library. The README describes how to build the code. The code itself runs LAMMPS on a subset of processors, sets up a LAMMPS problem by invoking LAMMPS input script commands one at a time, does a run, grabs atom coordinates, changes one atom position, puts them back into LAMMPS, and does another run.

This illustrates how an umbrella code could include new models and physics while using LAMMPS to perform MD, or how the umbrella code could call both LAMMPS and some other code to perform a coupled calculation.

6. 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 benchmark (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.

7. 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](#)
- [data2xmovie](#)
- [eam generate](#)
- [lmp2arc](#)
- [lmp2cfg](#)
- [lmp2traj](#)
- [lmp2vmd](#)
- [matlab](#)
- [micelle2d](#)
- [msi2lmp](#)
- [pymol_asphere](#)
- [python](#)
- [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](#).

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 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.

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

Imp2arc tool

The Imp2arc 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).

Imp2cfg tool

The Imp2cfg 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).

Imp2traj tool

The Imp2traj sub-directory contains a tool for converting LAMMPS output files into 3 analysis files. One file can be used to create contour maps of the atom positions over the course of the simulation. The other two files provide density profiles and dipole moments. See the README file for more information.

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

Imp2vmd tool

The Imp2vmd sub-directory contains some scripts for converting LAMMPS files into formats suitable for visualizing with the [VMD package](#).

One script extracts bond topology info from a LAMMPS data file. A second script attached amino acid (residue) information to what is extracted from a data file. See the README file for more information.

These 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 radial distribution output from the [fix rdf](#) command, 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, like

- 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 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.

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 or 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!

8. 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 adding two lines to the style_user.h file, in the same syntax as other LAMMPS classes are specified in the style.h file.

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
```

you put your 2 files in the LAMMPS src directory, add 2 lines to the style_user.h file, and re-make the code.

The first line added to style_user.h would be

```
PairStyle(foo,PairFoo)
```

in the #ifdef PairClass section, where "foo" is the style keyword in the pair_style command, and PairFoo is the class name in your C++ files.

The 2nd line added to style_user.h would be

```
#include "pair_foo.h"
```

in the #ifdef PairInclude section, where pair_foo.h is the name of your new include file.

When you re-make 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.

Here is a list of the new features that can be added in this way, along with information about how to submit your features for inclusion in the LAMMPS distribution.

- [Atom styles](#)
- [Bond, angle, dihedral, improper potentials](#)
- [Compute styles](#)
- [Dump styles](#)
- [Dump custom output options](#)
- [Fix styles](#) which include integrators, temperature and pressure control, force constraints, boundary conditions, diagnostic output, etc
- [Input script commands](#)
- [Kspace computations](#)
- [Minimization solvers](#)
- [Pairwise potentials](#)
- [Region styles](#)
- [Thermodynamic output options](#)
- [Variable options](#)

- [Submitting new features to the developers to include in LAMMPS](#)

As illustrated by the pairwise example, these 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 in these sections:

- [Dump custom output options](#)
- [Thermodynamic output options](#)
- [Variable options](#)

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.
-
-

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
unpack_comm	retrieve an atom's info from the buffer
pack_reverse	store an atom's info in a buffer communicating partial forces
unpack_reverse	retrieve an atom's info from the buffer
pack_border	store an atom's info in a buffer communicated on neighbor re-builds
unpack_border	retrieve an atom's info from the buffer
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. The atom arrays themselves are defined in atom.cpp. Search for the word "customize" and you will find locations you will need to modify.

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

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

Dump styles

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.

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 "never" which is used to determine when to call the 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.

Input script commands

New commands can be added to LAMMPS input scripts by adding new classes that have a "command" method and are listed in the Command sections of `style.h` (or `style_user.h`). 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.

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 `kpace.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

Minimization solvers

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

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.

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
-------	--------------------------------------------

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 several styles defined in `thermo.cpp`: "one", "multi", "granular", etc. 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.

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 = x123, y3, vx34, ... compute values = c_mytemp0, c_thermo_press3, ...

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::evaluate()` 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::evaluate()` 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::evaluate()` 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.

Submitting new features to the developers to include in LAMMPS

We encourage users to submit new features that they add to LAMMPS to [the developers](#), especially if you think they will be useful 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. Examples of user packages are in src sub-directories that start with USER. You can see a list of the both standard and user packages by typing "make package" in the LAMMPS src directory.

With user packages, 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 does requires changes in other LAMMPS files, you'll need to [communicate with the developers](#), since we may or may not want to make those changes.

Here is what you need to do to submit a user package for our consideration. Following these steps will save time for both you and us. See existing package files for examples.

Your user package will be a directory with a name like USER-FOO. In addition to your new files, the directory should contain a README, Install.csh and style_user_foo.h file. Send us a tarball of this USER-FOO directory.

The README text file should contain your name and contact information and a brief description of what your new package does.

The Install.csh and style_user_foo.h files enable LAMMPS to include and exclude your package.

Your new source files need to have the LAMMPS copyright, GPL notice, and your name at the top. 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 style and syntax as other LAMMPS files, though that would be nice.

Finally, in addition to the USER-FOO tarball, you also need to send us a documentation file for each new command or style you are adding to LAMMPS. These are text files which we will convert to HTML. Use one of the *.txt files in the doc dir as a starting point for the new file you create, since it should look similar to the doc files for existing commands and styles. The "Restrictions" section of the doc page should indicate that your feature is only available if LAMMPS is built with the "user-foo" package. See other user package files for an example of how to do this.

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).

9. Errors

This section describes the various kinds of errors you can encounter when using LAMMPS.

9.1 [Common problems](#)

9.2 [Reporting bugs](#)

9.3 [Error & warning messages](#)

9.1 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 different machines. Obviously, this means the phase space trajectories of the two simulations will rapidly diverge. See the discussion of the *loop* option in the [velocity](#) command for details.

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 mistakes, like choosing too big a timestep, specifying invalid force field coefficients, or putting 2 atoms on top of each other! If you find 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 exit gracefully when it encounters a fatal error. Sometimes it will print a WARNING and continue on; you can decide if the WARNING is important or not. 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.

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.

9.2 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](#).

9.3 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. If 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.

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 %d
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 use same angle style twice
Self-explanatory.

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 dump dcd
Self-explanatory.

Atom IDs must be consecutive for dump xtc
Self-explanatory.

Atom IDs must be consecutive for dump xyz
Self-explanatory.

Atom IDs must be consecutive for velocity create loop all
Self-explanatory.

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 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 command after simulation box is defined
The atom_modify command cannot be used after a read_data, read_restart, or create_box command.

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.

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 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.

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 %d

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 use same bond style twice

The sub-style arguments of bond_style hybrid cannot be duplicated. Check the input script.

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_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.

Cannot (yet) use PPPM with triclinic box
This feature is not yet supported.

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 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_box after simulation box is defined
The create_box command cannot be used after a read_data, read_restart, or create_box command.

Cannot displace_box on a non-periodic boundary
Self-explanatory.

Cannot dump scaled coords with triclinic box
Use dump custom with x,y,z instead.

Cannot find delete_bonds group ID
Group ID used in the delete_bonds command does not exist.

Cannot fix deform on a non-periodic boundary
Only a periodic boundary can be modified.

Cannot have both pair_modify shift and tail set to yes
These 2 options are contradictory.

Cannot open AIREBO potential file %s
Self-explanatory.

Cannot open EAM potential file %s
The specified EAM 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/spatial file %s

Self-explanatory.

Cannot open fix ave/time file %s

Self-explanatory.

Cannot open fix com file %s

The output file for the fix com command cannot be opened. Check that the path and name are correct.

Cannot open fix gyration file %s

Self-explanatory.

Cannot open fix msd file %s

The output file for the fix msd command 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 rdf file %s

The output file for the fix rdf 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 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 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 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 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 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 PPPM with 2d simulation

The kspace style ppm cannot be used in 2d simulations. You can use 2d PPPM in a 3d simulation; see the kspace_modify command.

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 dump bond 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 deform trapezoidal on a box with zero tilt
The trapezoidal style alters the current strain.

Cannot use fix enforce2d with 3d simulation
Self-explanatory.

Cannot use fix nph on a non-periodic dimension
Only periodic dimensions can be controlled with this fix.

Cannot use fix nph with triclinic box
This feature is not yet supported.

Cannot use fix nph without per-type mass defined
The defined atom style uses per-atom mass, not per-type mass.

Cannot use fix npt and fix deform on same dimension
These commands both change the box size/shape, so you cannot use both together.

Cannot use fix npt on a non-periodic dimension
Only periodic dimensions can be controlled with this fix.

Cannot use fix npt with triclinic box
This feature is not yet supported.

Cannot use fix npt without per-type mass defined
The defined atom style uses per-atom mass, not per-type mass.

Cannot use fix nvt without per-type mass defined
The defined atom style uses per-atom mass, not per-type mass.

Cannot use fix pour with triclinic box
This feature is not yet supported.

Cannot use fix press/berendsen and fix deform on same dimension
Self-explanatory.

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 press/berendsen without per-type mass defined
Self-explanatory.

Cannot use fix shake with non-molecular system
Your choice of atom style does not have bonds.

Cannot use fix temp/berendsen without per-type mass defined
Self-explanatory.

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 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 ppm, 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 tail corrections with 2d simulations
The correction factors are only currently defined for 3d systems.

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 velocity create loop all unless atoms have IDs
 Atoms in the simulation 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 reduce does not exist
 Self-explanatory.

Compute ID for fix ave/atom 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 must be alphanumeric or underscore characters
 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 erotate/asphere requires atom attribute shape
 An atom style that specifies atom shape is needed.

Compute erotate/asphere requires atom attributes angmom, quat
 An atom style is needed that has these attributes.

Compute erotate/sphere requires atom attribute omega
 An atom style is needed that has this attribute.

Compute erotate/sphere requires atom attribute shape
 An atom style that specifies atom shape is needed.

Compute erotate/sphere requires atom attributes radius, rmass
 An atom style is needed that has these attributes.

Compute erotate/sphere requires spherical particle shapes
 Self-explanatory.

Compute group/group group ID does not exist
 Self-explanatory.

Compute group/group requires pair style be defined
 Self-explanatory.

Compute in variable formula before initial run
 Calculating this compute before the first run is not allowed because various quantities may not yet be initialized.

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 temp ID does not compute temperature

The compute ID assigned to a pressure computation must compute temperature.
Compute reduce compute does not calculate a per-atom scalar
 Self-explanatory.

Compute reduce compute does not calculate a per-atom vector
 Self-explanatory.

Compute reduce compute does not calculate per-atom values
 Self-explanatory.

Compute reduce fix does not calculate a per-atom scalar
 Self-explanatory.

Compute reduce fix does not calculate a per-atom vector
 Self-explanatory.

Compute reduce fix does not calculate per-atom values
 Self-explanatory.

Compute reduce variable is not atom-style variable
 Self-explanatory.

Compute temp/asphere requires atom attributes quat, angmom
 The atom style defined does not have these attributes.

Compute temp/sphere requires atom attribute omega
 An atom style is needed that has this attributes.

Compute temp/sphere requires spherical particle shapes
 Self-explanatory.

Compute vector in variable formula is too small
 The index to the vector is out of bounds.

Computed temperature for fix temp/berendsen cannot be 0.0
 Self-explanatory.

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 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 group ID
 Self-explanatory.

Could not find compute pressure temp 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
 A group ID used in the displace_atoms command does not exist.

Could not find displace_box group ID
 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 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 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 spring couple 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 press ID
The compute ID for computing pressure does not exist.

Could not find fix_modify temp ID
The compute ID for computing temperature does not exist.

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 press ID
The compute ID needed by thermo style custom to compute pressure does not exist.

Could not find thermo_modify temp 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 temp ID
The compute ID needed by the velocity command to compute temperature does not exist.

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.

Could 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 must be of type inside
The region used in the create_box command must not be an "outside" region. See the region command for details.

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_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 > ghost cutoff
Cannot delete atoms further away than a processor knows about.

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.

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 %d
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 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.

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.

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 ID does not compute peratom info
The compute ID used must compute peratom info, not a global scalar or vector quantity.

Dump custom compute ID does not compute scalar per atom
The compute ID used must compute a single peratom datum.

Dump custom compute ID does not compute vector per atom
The compute ID used must compute a vector of peratom data.

Dump custom compute ID vector is not large enough
The compute ID vector of peratom data is not as large as is being accessed.

Dump custom fix ID does not compute peratom info
The specified fix does not calculate per-atom values.

Dump custom fix ID does not compute scalar per atom
The dump custom command is attempting to access a scalar quantity.

Dump custom fix ID does not compute vector per atom
The dump custom command is attempting to access a vector quantity.

Dump custom fix ID vector is not large enough
The dump custom command is attempting to access an out-of-range vector value.

Dump custom variable is not atom-style variable
Only atom-style variables generate per-atom quantities, needed for dump output.

Dump dcd cannot dump unwrapped coords with triclinic box

Cannot use unwrap option with non-orthogonal simulation box.

Dump dcd must use group all

Self-explanatory.

Dump dcd of non-matching # of atoms

Every snapshot written by dump dcd must contain the same # of atoms.

Dump xtc must use group all

Self-explanatory.

Dump xtc must use group all

Self-explanatory.

Dump_modify region ID does not exist

Self-explanatory.

Dumping an atom quantity that isn't allocated

The chosen atom style does not define the per-atom vector being dumped.

Empty brackets in variable formula

A value inside the brackets is required for this formula element.

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.

Final box dimension due to fix deform is < 0.0

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/spatial does not exist

Self-explanatory.

Fix ID for fix ave/time does not exist

Self-explanatory.

Fix ID must be alphanumeric or underscore characters

Self-explanatory.

Fix ave/atom compute does not calculate a per-atom scalar

A compute used by fix ave/atom must generate per-atom values.

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 compute vector is accessed out-of-range

The index for the vector is out of bounds.

Fix ave/atom fix does not calculate a per-atom scalar

A fix used by fix ave/atom must generate per-atom values.

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 fix vector is accessed out-of-range

The index for the vector is out of bounds.

Fix ave/atom variable is not atom-style variable

A variable used by fix ave/atom must generate per-atom values.

Fix ave/spatial compute does not calculate a per-atom scalar

A compute used by fix ave/spatial must generate per-atom values.

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 scalar

A fix used by fix ave/spatial must generate per-atom values.

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 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 vector is accessed out-of-range

The index for the vector is out of bounds.

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 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/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 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 ID does not exist

Self-explanatory

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/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 freeze requires atom attribute torque

The atom style defined does not have this attribute.

Fix heat group has no atoms

Self-explanatory.

Fix in variable formula before initial run

Calculating this fix before the first run is not allowed because various quantities may not yet be initialized.

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 msd group has no atoms

Cannot compute diffusion for no atoms.

Fix nph periods must be > 0.0

The time window for pressure relaxation must be > 0

Fix npt periods must be > 0.0

The time window for temperature or pressure relaxation must be > 0

Fix npt/sphere requires atom attribute shape

An atom style that specifies atom shape is needed.

Fix npt/sphere requires spherical particle shapes

Self-explanatory.

Fix nve/sphere requires atom attribute mu

An atom style with this attribute is needed.

Fix nve/sphere requires atom attribute shape

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 atom attributes radius, rmass

An atom style with these attributes is needed.

Fix nve/sphere requires spherical particle shapes

Self-explanatory.

Fix nvt period must be > 0.0

The time window for temperature relaxation must be > 0

Fix nvt/sphere requires atom attribute shape

An atom style with this attribute is needed.

Fix nvt/sphere requires atom attributes omega, torque

An atom style with these attributes is needed.

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 pour region ID does not exist
Self-explanatory.

Fix pour requires atom attributes radius, rmass
The atom style defined does not have these attributes.

Fix press/berendsen period must be > 0.0
Self-explanatory.

Fix rdf requires a pair style be defined
Cannot use the rdf fix unless a pair style with a cutoff has been defined.

Fix recenter group has no atoms
Self-explanatory.

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 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 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 vector in variable formula is too small
Index into vector is out of bounds.

Fix viscosity swap value must be positive
Self-explanatory.

Fix viscosity vtarget value must be positive
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_modify press ID does not compute pressure
The compute ID assigned to the fix must compute pressure.

Fix_modify temp 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.

Granular pair styles do not use pair_coeff settings
The pair_coeff command cannot be used with granular force fields.

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.

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 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 fix heat attempt

The velocity rescaling about to be performed by fix heat is invalid.

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 %d

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 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.

Improperly defined but no improper types
The data file header lists improper but no improper types.

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 MEAM potential file
Incorrect number of words per line in the potential file.

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.

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.

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 insert 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.
Invalid angle style
 The choice of angle style is unknown.
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 Improvers 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 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 fix rdf command
 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 bond style
 The choice of bond style is unknown.
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 index for fix shake
 Self-explanatory. Check the fix shake command in the input script.
Invalid coeffs for this angle style
 Cannot set class 2 coeffs in data file for this angle style.
Invalid coeffs for this dihedral style
 Cannot set class 2 coeffs in data file for this dihedral style.
Invalid coeffs for this improper style
 Cannot set class 2 coeffs in data file for this improper 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 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: Improvers
Atom style does not allow impropers.

Invalid data file section: MiddleBondTorsion Coeffs
Atom style does not allow dihedrals.

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 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 threshold operator
 Operator keyword used for threshold specification is not recognized.

Invalid fix ID in variable formula
 The fix is not recognized.

Invalid fix nph command for a 2d simulation
 Cannot use style xy, yz, or xz for a 2d simulation.

Invalid fix nph command pressure settings
 Pressure settings for different components must be the same if the components are coupled.

Invalid fix npt command for a 2d simulation
 Cannot use style xy, yz, or xz for a 2d simulation.

Invalid fix npt command pressure settings
 Pressure settings for different components must be the same if the components are coupled.

Invalid fix press/berendsen command for a 2d simulation
 Can only use xyz or aniso styles in 2d.

Invalid fix press/berendsen command pressure settings
 Pressure settings for different components must be the same if the components are coupled.

Invalid fix style used in compute displace/atom command
 Only a fix of style coord/original can be used with this compute.

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 communicate command
 Self-explanatory.

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 improper style
 The choice of improper style is unknown.

Invalid improper type in Improvers section of data file
 Improper type must be positive integer and within range of specified improper types.

Invalid index in variable formula
 The index between brackets is less than or equal to 0.

Invalid keyword in dump custom command
 One or more attribute keywords are not recognized.

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 math or group function in variable formula

The math or group function is not recognized.

Invalid natoms for dump dcd

Natoms is initially 0 which is not valid for the dump dcd style. Natoms must be constant for the duration of the simulation.

Invalid natoms for dump xtc

Natoms is initially 0 which is not valid for the dump xtc style.

Invalid natoms for dump xyz

Natoms is initially 0 which is not valid for the dump xyz style.

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 random number seed in set command

Random number seed must be > 0.

Invalid region style

The choice of region style is unknown.

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 shear direction for fix wall/gran

Self-explanatory.

Invalid style in pair_write command

Self-explanatory. Check the input script.

Invalid syntax in variable formula

Self-explanatory.

Invalid thermo keyword in variable formula

The keyword is not recognized.

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 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 $\mathbf{a}_1 \times \mathbf{a}_2 = \mathbf{a}_3$.

Lattice primitive vectors are collinear

The specified lattice primitive vectors do not form 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 in variable formula

Self-explanatory.

Lost atoms via displace_atoms: original %.15g current %.15g

The displace_atoms command lost one or more atoms.

Lost atoms via displace_box: original %.15g current %.15g

The displace_box command lost one or more atoms.

Lost atoms: original %.15g current %.15g

A thermodynamic computation has detected lost atoms.

MEAM library error %d

A call to the MEAM Fortran library returned an error.

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 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.

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 Improvers
The Atoms section of a data file must come before an Improvers 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 molecular atom style with fix rigid molecule
Self-explanatory.

Must use a z-axis cylinder with fix pour
The axis of the cylinder region used with the fix insert 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 efield
Self-explanatory.

Must use charged atom style with fix efield
The atom style being used does not allow atoms to have assigned charges. Hence it will not work with this fix which generates a force due to an E-field acting on charge.

Must use fix gravity with fix pour
Insertion of granular particles must be done under the influence of gravity.

Must use molecular atom style with neigh_modify exclude molecule
The atom style must define a molecule ID to use the exclude option.

Must use region with side = in with fix deposit
Self-explanatory.

Must use region with side = in with fix pour
Self-explanatory.

Must use special_bonds = 1,1,1 with bond style quartic
The settings for the special_bonds command must be set as indicated when using bond style quartic.

Needed topology not in data file
The header of the data file indicated that bonds or angles or dihedrals or improvers would be included, but they were not present.

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 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 granular

Self-explanatory.

Neighbor multi not yet enabled for rRESPA

Self-explanatory.

Neighbor page size must be ≥ 10 x the one atom setting

This is required to prevent wasting too much memory.

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 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 bonds allowed with this atom style

Self-explanatory. Check data file.

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 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 rigid bodies defined

The fix specification did not end up defining any rigid bodies.

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.

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.

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 stencil extends too far, reduce PPPM order
The grid points that atom charge are mapped to cannot extend further than one neighbor processor away.
Reducing the PPPM order via the kspace_modify command will reduce the stencil distance.

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 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
An atom style that stores this, like atom_style dpd, must be used.

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
The atom style defined does not have these attributes.

Pair granular requires atom attributes radius, omega, torque
The atom style defined does not have these attributes.

Pair granular requires ghost atoms store velocity
An atom style that stores this, like atom_style granular, must be used.

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 requires atom attributes torque, shape
Use a different atom style.

Pair lubricate requires mono-disperse particles
This is a current restriction of this pair style.

Pair lubricate requires spherical particles
This is a current restriction of this pair style.

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
Use a different atom style.

Pair style AIREBO requires atom IDs
Self-explanatory.

Pair style AIREBO requires newton pair on
Self-explanatory.

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 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 pair_write
The pair style does not have a single() function, so it can not be invoked by the pair_write command.

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 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 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_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.

Per-atom compute in equal-style variable formula
Equal-style variables cannot use per-atom quantities.

Per-atom fix in equal-style variable formula
Equal-style variables cannot use per-atom quantities.

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.

Potential with shear history requires newton pair off
Granular potentials that include shear history effects can only be run with a newton setting where pairwise newton is "off".

Power by 0 in variable formula
Self-explanatory.

Press ID for fix nph does not exist
The compute ID needed to compute pressure for the fix does not exist.

Press ID for fix npt does not exist
The compute ID needed to compute pressure for the fix does not exist.

Press ID for fix press/berendsen does not exist
The compute ID needed to compute pressure for the fix does not exist.

Press ID for thermo does not exist
The compute ID needed to compute pressure for thermodynamics does not exist.

Proc grid in z != 1 for 2d simulation
There cannot be more than 1 processor in the z dimension of a 2d simulation.

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.

Region intersect region ID does not exist
Self-explanatory.

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 wish 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.

Respa inner cutoffs are invalid
 The first cutoff must be \leq the second cutoff.

Respa levels must be ≥ 1
 Self-explanatory.

Respa middle cutoffs are invalid
 The first cutoff must be \leq 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.

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 command upto value is before current timestep
 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.

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 %d
 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 %d
 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 %d
 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.

Sqrt of negative in variable formula

Self-explanatory.

Substitution for illegal variable

Input script line contained a variable that could not be substituted for.

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.

Target T for fix npt cannot be 0.0

Self-explanatory.

Target T for fix nvt cannot be 0.0

Self-explanatory.

Temp ID for fix bond/swap does not exist

Self-explanatory.

Temp ID for fix nph does not exist

The compute ID needed to compute temperature for the fix does not exist.

Temp ID for fix npt does not exist

The compute ID needed to compute temperature for the fix does not exist.

Temp ID for fix nvt does not exist

The compute ID needed to compute temperature for the fix does not exist.

Temp ID for fix press/berendsen does not exist

Self-explanatory.

Temp ID for fix temp/berendsen does not exist

Self-explanatory.

Temp ID for fix temp/rescale does not exist

The compute ID needed to compute temperature for the fix does not exist.

Temper command before simulation box is defined

The temper command cannot be used before a read_data, read_restart, or create_box command.

Temperature for fix nvt/sllod does not have a bias

The specified compute must compute temperature with a bias.

Temperature region ID does not exist

The region ID specified in the temperature command does not exist.

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 ID does not compute scalar info

The specified compute ID does not compute a scalar quantity as requested.

Thermo compute ID does not compute vector info

The specified compute ID does not compute a vector quantity as requested.

Thermo compute ID vector is not large enough

The specified compute ID does not compute a large enough vector quantity for the requested index.

Thermo custom variable is not equal-style variable

Only equal-style variables can be output with thermodynamics, not atom-style variables.

Thermo fix ID does not compute scalar info

Only fixes that compute global values can be output with thermodynamics.

Thermo fix ID does not compute vector info

Only fixes that compute global values can be output with thermodynamics.

Thermo fix ID vector is not large enough

Index into vector is out of bounds.

Thermo keyword in variable formula before initial run

Calculating a thermo keyword before the first run is not allowed because various quantities may not yet be initialized.

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 press ID does not compute pressure

The specified compute ID does not compute pressure.

Thermo_modify temp 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.

Timestep must be ≥ 0

Specified timestep size is invalid.

Too big a problem to replicate with molecular atom style

Molecular problems cannot become bigger than 2^{31} atoms (or bonds, etc) when replicated, else the atom IDs and other quantities cannot be stored in 32-bit quantities.

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 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 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 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.

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.

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 \geq 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 section in data file: %s

The keyword for a section of the data file is not recognized by LAMMPS.

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.

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 recenter with undefined lattice

Must use lattice command with fix recenter 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.

Variable name for compute reduce does not exist

Self-explanatory.

Variable name for fix ave/atom 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 must be alphanumeric or underscore characters

Self-explanatory.

Variable uses compute via thermo keyword that thermo does not

A thermo keyword used in the variable requires a particular compute be invoked. However, the thermodynamic output does not use this compute, so it has not been initialized. Include the keyword in thermodynamic output.

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 temp ID does not compute temperature

The compute ID given to the velocity command must compute temperature.

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:

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. Its 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. Its interaction strength will be truncated to attempt to prevent the bond from blowing up.

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 wall/reflect should come after all other integration fixes

This is because other integration fixes may alter the coordinates of an atom, so you want to reflect it back into the box only after the other fixes have made their adjustments.

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 insert 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 coord/atom

It is not efficient to use compute coord/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 msd

It is not efficient to use fix msd 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.

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.

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 respa levels compute no forces

This is computationally inefficient.

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.

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 angle_style to restart file value

The angle style defined in the LAMMPS input script does not match that of the restart file.

Resetting bond_style to restart file value

The bond style defined in the LAMMPS input script does not match that of the restart file.

Resetting boundary settings to restart file values

The boundary settings defined in the LAMMPS input script do not match that of the restart file.

Resetting dihedral_style to restart file value

The dihedral style defined in the LAMMPS input script does not match that of the restart file.

Resetting dimension to restart file value

The dimension value defined in the LAMMPS input script does not match that of the restart file.

Resetting improper_style to restart file value

The improper style defined in the LAMMPS input script does not match that of the restart file.

Resetting newton bond to restart file value

The value of the newton setting for bonds defined in the LAMMPS input script does not match that of the restart file.

Resetting pair_style to restart file value

The pair style defined in the LAMMPS input script does not match that of the restart file.

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.

Resetting unit_style to restart file value

The unit style defined in the LAMMPS input script does not match that of the restart file.

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 newton pair setting

The restart file was written out by a LAMMPS simulation running with a different value of the newton pair setting. The new simulation will use the value from the input script.

Restart file version does not match LAMMPS version

The version of LAMMPS that wrote the restart file does not match the version of LAMMPS that is reading the restart file. Generally this shouldn't be a problem, since restart file formats won't change very often if at all. But if they do, the code will probably crash trying to read the file. Versions of LAMMPS are specified by a date.

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.

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 NPH is not for group all

User-assigned temperature to NPH fix does not compute temperature for all atoms. Since NPH computes a global pressure, the kinetic energy contribution from the temperature is assumed to also be for all atoms.

Thus the pressure used by NPH 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 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.

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

Self-explanatory.

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.

10. 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.

10.1 [Coming attractions](#)

10.2 [Past versions](#)

10.1 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!

- NPT with changing box shape (Parinello–Rahman)
 - bond creation potentials
 - long–range point dipole solver
 - torsional shear boundary conditions and temperature calculation
 - charge equilibration
 - ReaxFF force field from Bill Goddard's group
-

10.2 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 , θ_0 , K_{ub} , and r_{ub} 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/radian²)
- θ_0 (degrees)
- K_{ub} (energy/distance²)
- r_{ub} (distance)

θ_0 is specified in degrees, but LAMMPS converts it to radians internally; hence the units of K are in energy/radian².

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
```

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 E_a is the angle term, E_{bb} is a bond–bond term, and E_{ba} is a bond–angle term. θ_0 is the equilibrium angle and r_1 and r_2 are the equilibrium bond lengths.

See [\(Sun\)](#) for a description of the COMPASS *class2* force field.

For this style, coefficients for the E_a formula can be specified in the input script or data file. These are the 4 coefficients:

- θ_0 (degrees)
- K_2 (energy/radian²)
- K_3 (energy/radian³)
- K_4 (energy/radian⁴)

θ_0 is specified in degrees, but LAMMPS converts it to radians internally; hence the units of the various K are in per–radian.

Coefficients for the E_{bb} and E_{ba} formulas can only be specified in the data file.

For the E_{bb} formula, the coefficients are listed under a "BondBond Coeffs" heading and each line lists 3 coefficients:

- M (energy/distance²)
- r_1 (distance)
- r_2 (distance)

For the E_{ba} formula, the coefficients are listed under a "BondAngle Coeffs" heading and each line lists 4 coefficients:

- N1 (energy/distance²)
- N2 (energy/distance²)
- 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 θ_0 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/radian²)
- θ_0 (degrees)
- *cg_type* (string, one of *lj9_6*, *lj12_4*, *lj12_6*)
- *epsilon* (energy units)
- *sigma* (distance units)

θ_0 is specified in degrees, but LAMMPS converts it to radians internally; hence the units of K are in energy/radian².

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
```

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/squared](#) – cosine squared angle potential

- [angle_style harmonic](#) – harmonic angle
-

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 θ_0 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)
- θ_0 (degrees)

θ_0 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/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 1.2
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` command, the first coefficient sets the angle style and the remaining coefficients are those appropriate to that style. In the example above, the 2 `angle_coeff` commands would set angles of angle type 1 to be computed with a *harmonic* potential with coefficients 80.0, 1.2 for K, r0. All other angle types (2–N) would be computed with a *cosine* potential with coefficient 50.0 for K.

If the angle *class2* potential is one of the hybrid styles, it requires additional BondBond and BondAngle coefficients be specified in the data file. These lines must also have an additional "class2" argument added after the angle type. For angle types which are assigned to other hybrid styles, use the style name (e.g. "harmonic") appropriate to that style. The BondBond and BondAngle coeffs for that angle type will then be ignored.

An angle style of *none* can be specified as the 2nd argument to the `angle_coeff` command, if you desire to turn off certain 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/squared](#) – cosine squared angle potential
 - [angle_style harmonic](#) – harmonic angle
-

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`

atom_modify command

Syntax:

```
atom_modify keyword value ...
```

- one or more keyword/value pairs may be appended
- keyword = *map* or *first*

map value = *array* or *hash*

first value = group-ID = group whose atoms will appear first in internal atom lists

Examples:

```
atom_modify map hash
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 is 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 because the `atom_modify` command must be used in an input script before a simulation is setup and groups are defined, this means the *first* keyword will specify a group that is not yet defined. This is OK; LAMMPS will check that the group exists before the first simulation is performed.

Restrictions:

This command must be used before the simulation box is defined by a [read_data](#) or [create_box](#) command.

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, `first = all`.

atom_style command

Syntax:

```
atom_style style args
```

- *style* = *angle* or *atomic* or *bond* or *charge* or *dipole* or *dpd* or *ellipsoid* or *full* or *granular* or *molecular* or *peri* 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 attributes of each style. All styles store coordinates, velocities, atom IDs and types.

- *angle* = bonds and angles – e.g. bead-spring polymers with stiffness
- *atomic* = only the default values
- *bond* = bonds – e.g. bead-spring polymers
- *charge* = charge
- *dipole* = charge and dipole moment
- *dpd* = default values, also communicates velocities
- *ellipsoid* = quaternion for particle orientation, angular velocity/momentum
- *full* = molecular + charge – e.g. biomolecules, charged polymers
- *granular* = granular atoms with rotational properties
- *molecular* = bonds, angles, dihedrals, impropers – e.g. all-atom polymers
- *peri* = mass, volume – e.g. mesoscopic Peridynamics

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. E.g. if you want charged DPD particles,

you would need to use "atom_style hybrid dpd charge". 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 *granular* style is part of the "granular" package. The *dpd* style is part of the "dpd" package. The *dipole* style is part of the "dipole" package. The *ellipsoid* style is part of the "ellipsoid" package. The *peri* style is part of the "peri" package for Peridynamics. 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 r_0 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:

- R_0 (distance)
- K_2 (energy/distance²)
- K_3 (energy/distance³)
- K_4 (energy/distance⁴)

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
-

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

$$E = -0.5K R_0^2 \ln \left[1 - \left(\frac{r}{R_0} \right)^2 \right] + 4\epsilon \left[\left(\frac{\sigma}{r} \right)^{12} - \left(\frac{\sigma}{r} \right)^6 \right] + \epsilon$$

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 R_0 , 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/distance²)
- R_0 (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 0 1 1](#) to use this bond style. LAMMPS will issue a warning if 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 $R_0 + \text{delta}$ and the 2nd term is cutoff at $2^{1/6} \sigma + \text{delta}$.

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 0 1 1](#) to use this bond style. LAMMPS will issue a warning if 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 r_0 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/distance²)
- r_0 (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 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` command, the first coefficient sets the bond style and the remaining coefficients are those appropriate to that style. In the example above, the 2 `bond_coeff` commands would 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) would be computed with a *fene* potential with coefficients 30.0, 1.5, 1.0, 1.0 for K, R0, epsilon, sigma.

A bond style of *none* can be specified as the 2nd argument to the `bond_coeff` command, if you desire to turn off certain 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 restarting 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

$$E = D \left[1 - e^{-\alpha(r-r_0)} \right]^2$$

where r_0 is the equilibrium bond distance, α 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)
- α (inverse distance)
- r_0 (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 r_0 and maximum extension λ .

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)
- r_0 (distance)
- λ (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 [dump bond](#), 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
-

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`

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 = *group* or *cutoff*

```
group value = group-ID = only communicate atoms in the group
cutoff value = Rcut (distance units) = communicate atoms from this far away
```

Examples:

```
communicate multi
communicate multi group solvent
communicate single cutoff 5.0
```

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 *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 *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.

Restrictions: none

Related commands:

[neighbor](#)

Default:

The default settings are style = single, group = all, cutoff = 0.0. The cutoff default of 0.0 means that effectively 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:

Create 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. There are two kinds of computes, "global" computes that calculate one or more values for the entire group of atoms, and "per-atom" computes that calculate one or more values for each atom in the group. The latter has the word "atom" in its style name.

In LAMMPS, a "compute" can be used in several ways. The results of global computes 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. The results of computes that calculate a global temperature or pressure can be used by fixes that do thermostating or barostating and when atom velocities are created.

The results of per-atom computes can be output via the [dump custom](#) command or the [fix ave/spatial](#) command. Or the per-atom values can be time-averaged via the [fix ave/atom](#) command and then output via the [dump custom](#) or [fix ave/spatial](#) commands. Or the per-atom values can be referenced in a [variable atom](#) command. Note that the value of per-atom computes will be 0.0 for atoms not in the specified compute group.

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.

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 depending on the "thermo_modify norm" setting. But if a compute value is accessed in another way, e.g. by a [variable](#), you may need to know whether it is an intensive or extensive value. See the doc page for individual computes for further info.

LAMMPS creates its own global computes 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 thermostating or barostating, may also create computes. These are discussed in the documentation for specific [fix](#) commands.

In all these cases, the default computes 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:

- [centro/atom](#) – centro-symmetry parameter for each atom
- [coord/atom](#) – coordination number for each atom
- [damage/atom](#) – Peridynamic damage for each atom
- [displace/atom](#) – displacement of each atom
- [erotate/asphere](#) – rotational energy of aspherical particles
- [erotate/sphere](#) – rotational energy of spherical particles
- [group/group](#) – energy/force between two groups of atoms
- [ke](#) – translational kinetic energy
- [ke/atom](#) – kinetic energy for each atom
- [pe](#) – potential energy
- [pe/atom](#) – potential energy for each atom
- [pressure](#) – total pressure and pressure tensor
- [reduce](#) – combine per-atom quantities into a single global value
- [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/dipole](#) – temperature of point dipolar particles
- [temp/partial](#) – temperature excluding one or more dimensions of velocity
- [temp/ramp](#) – temperature excluding ramped velocity component
- [temp/region](#) – temperature of a region of atoms
- [temp/sphere](#) – temperature of spherical particles

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](#).

Restrictions: none

Related commands:

[uncompute](#), [compute_modify](#), [fix ave/atom](#), [fix ave/spatial](#), [fix ave/time](#)

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 computes as input. See [this section](#) for an overview of LAMMPS output options.

Restrictions:

This compute is part of the "user-ackland" 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 centro/atom command

Syntax:

```
compute ID group-ID centro/atom
```

- ID, group-ID are documented in [compute](#) command
- centro/atom = style name of this compute command

Examples:

```
compute 1 all centro/atom
```

Description:

Define a computation that calculates the centro-symmetry parameter for each atom in a 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 [\(Kelchner\)](#)

$$P = \sum_{i=1}^6 |\vec{R}_i + \vec{R}_{i+6}|^2$$

where the 12 nearest neighbors are found (for fcc lattices) and R_i and R_{i+6} are the vectors from the central atom to the opposite pair of nearest neighbors. Atoms not in the group are included in the 12 neighbors used in this calculation.

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 *centro/atom* style.

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.

Restrictions: none

Related commands: none

Default: none

(Kelchner) Kelchner, Plimpton, Hamilton, Phys Rev B, 58, 11085 (1998).

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 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.

Restrictions: none

Related commands: none

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 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.

Restrictions:

The *damage/atom* style 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 displace/atom command

Syntax:

```
compute ID group-ID displace/atom fix-ID
```

- ID, group-ID are documented in [compute](#) command
- displace/atom = style name of this compute command
- fix-ID = ID of a fix of style [coord/original](#)

Examples:

```
compute 1 all displace/atom myCoords
```

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. The original coordinates are stored by a [fix coord/original](#) command, whose ID must be specified with this compute. They are the coordinates at the time the fix command was issued. The displacement can be output directly via the [dump custom](#) command. It could also be used to calculate a diffusion coefficient for a group of atoms, as is done by the [fix msd](#) command.

The value of the displacement will be 0.0 for atoms not in the specified compute group. Note that the fix defines its own group and stores (0.0,0.0,0.0) for the original coordinates of any atom not in the fix group.

Output info:

This compute calculates a vector of length 4 for each atom, which can be accessed by indices 1–4 by any command that uses per-atom computes as input. See [this section](#) for an overview of LAMMPS output options. The first 3 components of the vector are the x,y,z displacements. The 4th component is the total displacement, i.e. $\sqrt{dx^2 + dy^2 + dz^2}$.

Restrictions: none

Related commands:

[dump custom](#), [fix msd](#)

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:

The scalar value calculated by this compute is "extensive", meaning it scales with the number of atoms in the simulation.

Restrictions:

This compute requires that particles be represented as extended ellipsoids and not point particles. This means they will have an angular momentum and a shape which is determined by the [shape](#) command.

Related commands: none

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 $\frac{1}{2} I w^2$, 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:

The scalar value calculated by this compute is "extensive", meaning it scales with the number of atoms in the simulation.

Restrictions:

This compute requires that particles be represented as extended spheres and not point particles. This means they will have an angular velocity and a diameter which is determined either by the [shape](#) command or by each particle being assigned an individual radius, e.g. for [atom_style granular](#).

Related commands: none

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:

The interaction energy calculated by this compute is a scalar quantity. The interaction force is a vector of length 3. Both the scalar and vector values calculated by this compute are "extensive", meaning they scale with the number of atoms in the simulation.

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 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 of each particle is computed as $\frac{1}{2} m v^2$, 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 $\frac{1}{2} k_B T$ 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:

The scalar value calculated by this compute is "extensive", meaning it scales with the number of atoms in the simulation.

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 $\frac{1}{2} m v^2$, 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 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.

Restrictions: none

Related commands:

[dump custom](#)

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.

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. 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 pe 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 *kpace*

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.

Output info:

The scalar value calculated by this compute is "extensive", meaning it scales with the number of atoms in the simulation.

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 energies can be accessed as scalar values by any command that uses per-atom computes, e.g. the [dump custom](#) command or [fix ave/spatial](#) command or [fix ave/atom](#) command. See [this section](#) for an overview.

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          p all reduce sum c_peratom
thermo_style      custom step temp etotal press pe c_p
```

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.

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.

Restrictions:

Related commands:

[compute pe](#), [compute stress/atom](#)

Default: none

compute pressure 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 *kpace* or *fix*

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

$$P = \frac{Nk_B T}{V} + \frac{\sum_i^N \mathbf{r}_i \cdot \mathbf{f}_i}{dV}$$

where N is the number of atoms in the system (see discussion of DOF below), K_b 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 6-component pressure tensor is also calculated by this compute whose components can be output by the [thermo_style custom](#) command or accessed by other [compute](#) and [fix](#) commands. The equation for the components of the tensor is the same as in above formula, except that the first term uses the components of the kinetic energy tensor (v_x * v_y instead of v² for temperature) and the second term uses R_x * F_y for the W_{xy} component of the virial tensor, etc.

If no extra keywords are listed, the entire equation above is calculated which includes a temperature term (or kinetic energy tensor) and the virial as the sum of pair, bond, angle, dihedral, improper, kspace (long-range), and fix contributions. If any extra keywords are listed, then only those components are summed to compute ke and/or the virial.

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 above formula is really degrees-of-freedom where the DOF is specified 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.

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: none

Related commands:

[compute temp](#), [compute stress/atom](#), [thermo_style](#),

Default: none

compute reduce command

Syntax:

```
compute ID group-ID reduce mode input1 input2 ...
```

- ID, group-ID are documented in [compute](#) command
- reduce = style name of this compute command
- mode = *sum* or *min* or *max*
- 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 scalar value calculated by a compute with ID
c_ID[N] = Nth component of per-atom vector calculated by a compute with ID
f_ID = per-atom scalar value calculated by a fix with ID
f_ID[N] = Nth component of per-atom vector calculated by a fix with ID
v_name = per-atom value calculated by an atom-style variable with name
```

Examples:

```
compute 1 all reduce sum c_force
compute 2 all reduce min c_press2 f_ave v_myKE
```

Description:

Define a calculation that "reduces" one or more per-atom inputs across all atoms in the group to yield a single global scalar for each listed input. The resulting value(s) can be accessed by any command that uses global computes, e.g. the [thermo custom](#) command or [fix ave/time](#) command or by a [variable](#) command. See [this section](#) of the documentation for an overview of output options.

The reduction operation is specified by the *mode* setting. The *sum* option adds the per-atom quantities into a global total. The *min* or *max* options find the minimum or maximum value across all per-atom quantities.

Each listed input is operated on independently. The group specified with the command means only atoms within the group contribute to the result. Note that the input that produces the per-atom quantities may define its own group which affects the quantities it returns. For example, if a per-atom compute is used as an input, 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](#). In the latter cases, the compute, fix, or variable must produce per-atom quantities, not a global quantity.

[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 compute since all other variable styles produce global quantities.

If a single input is specified this compute produces a global scalar value. If multiple inputs are specified, this compute produces a vector of global 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:

For *sum* mode, the scalar and vector values calculated by this compute are "extensive", meaning they scale with the number of atoms in the simulation. For *min* and *max* modes, the value(s) are intensive.

Restrictions: none

Related commands:

`compute`, `fix`, `variable`

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*

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: xx, yy, zz, xy, xz, yz. See the [compute pressure](#) command if you want the stress tensor (pressure) of the entire system. The 6 components can be accessed by indices 1–6 by any command that uses per-atom computes as input, e.g. the [dump custom](#) command or [fix ave/spatial](#) command or [fix ave/atom](#) command. See [this section](#) for an overview.

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:

$$S_{ab} = - \left[mv_a v_b + \frac{1}{2} \sum_{n=1}^{N_p} (r_{1a} F_{1b} + r_{2a} F_{2b}) + \frac{1}{2} \sum_{n=1}^{N_b} (r_{1a} F_{1b} + r_{2a} F_{2b}) + \right. \\ \left. \frac{1}{3} \sum_{n=1}^{N_a} (r_{1a} F_{1b} + r_{2a} F_{2b} + r_{3a} F_{3b}) + \frac{1}{4} \sum_{n=1}^{N_d} (r_{1a} F_{1b} + r_{2a} F_{2b} + r_{3a} F_{3b} + r_{4a} F_{4b}) + \right. \\ \left. \frac{1}{4} \sum_{n=1}^{N_i} (r_{1a} F_{1b} + r_{2a} F_{2b} + r_{3a} F_{3b} + r_{4a} F_{4b}) + \sum_{n=1}^{N_f} r_{ia} F_{ib} \right]$$

The first term is a kinetic energy contribution for atom I . The second term is a pairwise energy contribution where n loops over the N_p 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 N_b bonds which atom I is part of. There are similar terms for the N_a angle, N_d dihedral, and N_i improper interactions atom I is part of. Finally, there is a term for the N_f [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 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.

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 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 $3V$, where 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 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 vector of length 6 for each atom, which can be accessed by indices 1–6 by any command that uses per-atom computes as input. See [this section](#) for an overview of LAMMPS output options. The 6 components of the vector are ordered xx, yy, zz, xy, xz, yz.

Restrictions: none

Related commands:

[compute pe](#), [compute stress/atom](#)

Default: none

compute temp 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 $KE = \frac{dim}{2} N k T$, where KE = total kinetic energy of the group of atoms (sum of $\frac{1}{2} m v^2$), dim = 2 or 3 = dimensionality of the simulation, N = number of atoms in the group, k = Boltzmann constant, and T = temperature.

A 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 $v_x * v_y$ for the xy component, etc.

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 thermostating.

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: none

Related commands:

[compute temp/partial](#), [compute temp/region](#), [compute pressure](#)

Default: none

compute temp/asphere command

Syntax:

```
compute ID group-ID temp/asphere bias-ID
```

- ID, group-ID are documented in [compute](#) command
- temp/asphere = style name of this compute command
- bias-ID = ID of a temperature compute that removes a velocity bias (optional)

Examples:

```
compute 1 all temp/asphere  
compute myTemp mobile temp/asphere tempCOM
```

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.

For 3d aspherical particles, each has 3, 5, or 6 degrees of freedom (3 translational, remainder rotational), depending on whether the particle is spherical, uniaxial, or biaxial. This is determined by the [shape](#) command. Uniaxial means two of its three shape parameters are equal. Biaxial means all 3 shape parameters are different.

For 2d aspherical particles, each has 3 or 4 degrees of freedom (3 translational, remainder rotational), depending on whether the particle is spherical, or biaxial. Biaxial means the x,y shape parameters are unequal.

IMPORTANT NOTE: These degrees of freedom assume that the interaction potential between degenerate aspherical particles does not impart rotational motion to the extra degrees of freedom. E.g. the [GayBerne pair potential](#) does not impart torque to spherical particles, so they do not rotate.

The rotational kinetic energy is computed as $\frac{1}{2} I w^2$, 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 6-component kinetic energy tensor 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 v_x*v_y and w_x*w_y for the xy component, and the appropriate elements of the inertia tensor are used.

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.

If a *bias-ID* is specified it must be 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.

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.

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 requires that particles be represented as extended ellipsoids and not point particles. This means they will have an angular momentum and a shape which is determined by the [shape](#) command.

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.

The temperature is calculated by the formula $KE = \text{dim}/2 N k T$, 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 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 $v_x * v_y$ for the xy component, etc.

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 thermostating then this bias will be subtracted from each atom, thermostating of the remaining thermal velocity will be performed, and the bias will be added back in. Thermostating 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 thermostating.

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: 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 thermostating 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 point 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 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. 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.

The temperature is calculated by the formula $KE = \text{dim}/2 N k T$, 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 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 $v_x * v_y$ for the xy component, etc.

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 thermostating then this bias will be subtracted from each atom, thermostating 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:

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: none

Related commands:

[compute temp/ramp](#), [fix deform](#), [fix nvt/sllod](#)

Default: none

compute temp/partial 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 compute to calculate 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 $KE = \text{dim}/2 N k T$, 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 6-component kinetic energy tensor 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 $v_x * v_y$ for the xy component, etc.

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 thermostating then this bias will be subtracted from each atom, thermostating of the remaining thermal velocity will be performed, and the bias will be added back in. Thermostating 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 thermostating.

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: none

Related commands:

[compute temp](#), [compute temp/region](#), [compute pressure](#)

Default: none

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:

```
temperature 2nd middle 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 imposed velocity on the system 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 is the same as for the [velocity ramp](#) command which was presumably used to impose the velocity.

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 6-component kinetic energy tensor 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 $v_x * v_y$ for the xy component, etc.

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 thermostating then this bias will be subtracted from each atom, thermostating of the remaining thermal velocity will be performed, and the bias will be added back in. Thermostating 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 thermostating.

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: none

Related commands:

[compute temp](#), [compute temp/region](#), [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:

```
temperature mine flow region boundary
```

Description:

Define a computation that calculates the temperature of a group of atoms in a geometric region. This can be useful for thermostating 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 npb](#)), if the degrees-of-freedom included in the computed T varies with time.

The temperature is calculated by the formula $KE = \text{dim}/2 N k T$, 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 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 $v_x * v_y$ for the xy component, etc.

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 thermostating then this bias will be subtracted from each atom, thermostating of the remaining thermal velocity will be performed, and the bias will be added back in. Thermostating fixes that work in this way include [fix nvt](#), [fix temp/rescale](#), [fix temp/berendsen](#), and [fix langevin](#). This means any of the thermostating 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 thermostating.

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: none

Related commands:

[compute temp](#), [compute pressure](#)

Default: none

compute temp/sphere command

Syntax:

```
compute ID group-ID temp/sphere bias-ID
```

- ID, group-ID are documented in [compute](#) command
- temp/sphere = style name of this compute command
- bias-ID = ID of a temperature compute that removes a velocity bias (optional)

Examples:

```
compute 1 all temp/sphere  
compute myTemp mobile temp/sphere tempCOM
```

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.

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).

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 6-component kinetic energy tensor 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 v_x*v_y and w_x*w_y for the xy component.

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.

If a *bias-ID* is specified it must be 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 thermostating for more details.

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 thermostating.

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 requires that particles be represented as extended spheres and not point particles. This means they will have an angular velocity and a diameter which is determined either by the [shape](#) command or by each particle being assigned an individual radius, e.g. for [atom_style granular](#).

Related commands:

[compute temp](#), [compute temp/asphere](#)

Default: none

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*

```
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)
```

- 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 as an alternative to reading in their coordinates 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 exception is for the *single* style with units = box.

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.

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.

Aside from their position and atom type, other properties of created atoms are set to 0.0, e.g. velocity, charge, etc. These properties can be changed via the [velocity](#) or [set](#) commands.

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.

Restrictions:

An [atom_style](#) must be previously defined to use this command.

Related commands:

[lattice](#), [region](#), [create_box](#), [read_data](#), [read_restart](#)

Default: none

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) 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. See the [region prism](#) command for a description of how the shape of the parallelepiped is defined. The parallelepiped has its "origin" at (xlo,ylo,zlo) and 3 edge vectors starting from its origin given by $a = (xhi-xlo,0,0)$; $b = (xy,yhi-ylo,0)$; $c = (xz,yz,zhi-zlo)$.

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 equivalent.

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.

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
```

- 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)
```

Examples:

```
delete_atoms group edge
delete_atoms region sphere
delete_atoms overlap 0.3 all all
delete_atoms overlap 0.5 solvent colloid
delete_atoms porosity cube 0.1
```

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.

After atoms are deleted, if the system is not molecular (no bonds), 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, 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: none

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
```

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 E_d is the dihedral term, E_{mbt} is a middle-bond-torsion term, E_{ebt} is an end-bond-torsion term, E_{at} is an angle-torsion term, E_{aat} is an angle-angle-torsion term, and E_{bb13} 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.

For this style, coefficients for the E_d formula can be specified in either the input script or data file. These are the 6 coefficients:

- K1 (energy)
- phi1 (degrees)
- K2 (energy)
- phi2 (degrees)
- K3 (energy)
- phi3 (degrees)

Coefficients for all the other formulas can only be specified in the data file.

For the E_{mbt} formula, the coefficients are listed under a "MiddleBondTorsion Coeffs" heading and each line lists 4 coefficients:

- A1 (energy/distance)
- A2 (energy/distance)
- A3 (energy/distance)
- r2 (distance)

For the Eebt formula, the coefficients are listed under a "EndBondTorsion Coeffs" heading and each line lists 8 coefficients:

- 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, the coefficients are listed under a "AngleTorsion Coeffs" heading and each line lists 8 coefficients:

- 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, the coefficients are listed under a "AngleAngleTorsion Coeffs" heading and each line lists 3 coefficients:

- M (energy/radian²)
- 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/radian².

For the Ebb13 formula, the coefficients are listed under a "BondBond13 Coeffs" heading and each line lists 3 coefficients:

- N (energy/distance²)
- 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
```

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

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 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, dipoles in a polymer flow (of dihedral type 1) could be computed with a *harmonic* potential and dipoles 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` command, the first coefficient sets the dihedral style and the remaining coefficients are those appropriate to that style. In the example above, the 2 `dihedral_coeff` commands would set dipoles of dihedral type 1 to be computed with a *harmonic* potential with coefficients 80.0, 1.2 for K, d, n. Dihedral type 2 would be computed with a *helix* potential with coefficients 10.0, 10.0, 10.0 for A, B, C.

If the dihedral *class2* potential is one of the hybrid styles, it requires additional MiddleBondTorsion, EndBondTorsion, AngleTorsion, AngleAngleTorsion, and BondBond13 coefficients be specified in the data file. These lines must also have an additional "class2" argument added after the dihedral type. For dihedral types which are assigned to other hybrid styles, use the style name (e.g. "harmonic") appropriate to that style. The MiddleBondTorsion, etc coeffs for that dihedral type will then be ignored.

A dihedral style of *none* can be specified as the 2nd argument to the `dihedral_coeff` command, if you desire to turn off certain 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 restarting 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 opl](#) – OPLS dihedral
-

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
```

dipole command

Syntax:

```
dipole I value
```

- I = atom type (see asterisk form below)
- value = dipole moment (dipole units)

Examples:

```
dipole 1 1.0  
dipole 3 2.0  
dipole 3*5 0.0
```

Description:

Set the dipole moment for all atoms of one or more atom types. This command is only used for atom styles that require dipole moments ([atom_style dipole](#)). A value of 0.0 should be used if the atom type has no dipole moment. Dipole values can also be set in the [read_data](#) data file. See the [units](#) command for a discussion of dipole units.

Currently, only [atom_style dipole](#) requires dipole moments be set.

I 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 dipole moment 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 specifies a dipole moment uses the same format as the arguments of the dipole command in an input script, except that no wild-card asterisk can be used. For example, under the "Dipoles" section of a data file, the line that corresponds to the 1st example above would be listed as

```
1 1.0
```

Restrictions:

This command must come after the simulation box is defined by a [read_data](#), [read_restart](#), or [create_box](#) command.

All dipoles moments must be defined before a simulation is run (if the atom style requires dipoles be set). They must also all be defined before a [set dipole](#) or [set dipole/random](#) command is used.

Related commands: none

Default: none

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

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 *bond* or *dcd* or *xtc* or *xyz* 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
bond args = none
dcd args = none
xtc args = none
xyz args = none
custom args = list of atom attributes
    possible attributes = id, mol, type,
                        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,
                        quatw, quati, quatj, quatk, tqx, tqy, tqz,
                        c_ID, c_ID[N], f_ID, f_ID[N], v_name

id = atom ID
mol = molecule ID
type = atom type
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
quatw,quati,quatj,quatk = quaternion components for aspherical particles
tqx,tqy,tqz = torque on extended particles
c_ID = per-atom scalar value calculated by a compute with ID
c_ID[N] = Nth component of per-atom vector calculated by a compute with ID
f_ID = per-atom scalar value calculated by a fix with ID
f_ID[N] = Nth component of per-atom vector calculated by a fix with ID
v_name = per-atom value 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 1 all xtc 1000 file.xtc
```

Description:

Dump a snapshot of atom quantities to one or more files every N timesteps in one of several styles. 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). Only information for atoms in the specified group is dumped. The [dump_modify](#) command can also alter what atoms are included. Not all styles support all these options; see details below.

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.

When LAMMPS is running in parallel, the atom information written to dump files (typically one line per atom) may be written in an indeterminate order. This is because data for a single snapshot is collected from multiple processors. This is always the case for the *atom*, *bond*, and *custom* styles. It is also the case for the *xyz* style if the dump group is not *all*. It is not the case for the *dcd* and *xtc* styles which always write atoms in sorted order. So does the *xyz* style if the dump group is *all*.

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*, *bond*, 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 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](#).

For style *bond*, the bond topology between atoms is written, in the same format specified in data files read in by the [read_data](#) command. Both atoms in the bond must be in the dump group for the bond to be written. Any bonds that have been broken (see the [bond_style](#) command) by setting their bond type to 0 are not written. 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.

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 dump custom quantities is given below.

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 dump group must be *all* for 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 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 *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 dump group must be *all* for 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). We are told VMD will also read LAMMPS *atom* style dump files since someone has added a LAMMPS format plug-in to VMD. It may require an initial snapshot from an XYZ formatted file to get started.

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. N can be changed between runs by using the [dump_modify](#) 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 wild-card 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.

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 atom quantities that can be specified as part of the *custom* style.

The *id*, *mol*, *type*, *x*, *y*, *z*, *vx*, *vy*, *vz*, *fx*, *fy*, *fz*, *q* keywords are self-explanatory. *Id* is the atom ID. *Mol* is the molecule ID, included in the data file for molecular systems. The *x*, *y*, *z* keywords 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. 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 size printed with the snapshot. The image flags can be printed directly using the *ix*, *iy*, *iz* keywords. The [dump_modify](#) command describes in more detail what is meant by scaled vs unscaled coordinates and the image flags.

The *mux*, *muy*, *muz* keywords are specific to dipolar systems defined with an atom style of *dipole*. They give the orientation of the atom's point dipole moment.

The *radius* keyword is specific to extended spherical particles that have a finite size, such as granular particles defined with an atom style of *granular*.

The *omegax*, *omegay*, and *omegaz* keywords are specific to extended spherical or aspherical particles that have an angular velocity. Only certain atom styles, such as *granular* or *dipole* define this quantity.

The *angmomx*, *angmomy*, and *angmomz* keywords are specific to extended aspherical particles that have an angular momentum. Only the *ellipsoid* atom style defines this quantity.

The *quatw*, *quati*, *quatj*, *quatk* keywords are for aspherical particles defined with an atom style of *ellipsoid*. They are the components of the quaternion that defines the orientation of the particle.

The *txx*, *txy*, *txz* keywords are for extended spherical or aspherical particles that can sustain a rotational torque due to interactions with other particles.

The *c_ID* and *c_ID[N]* keywords allow scalar or vector per-atom quantities calculated by a [compute](#) to be output. The ID in the keyword 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 per-atom computes for calculating the energy, stress, centro-symmetry parameter, and coordination number of individual atoms.

Note that computes which calculate global scalar and vector quantities, as opposed to per-atom quantities, cannot be output in a dump. Instead, these quantities can be output by the [thermo_style custom](#) command.

If *c_ID* is used as a keyword, then the scalar per-atom quantity 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 component of the M-length per-atom vector calculated by the compute.

The *f_ID* and *f_ID[N]* keywords allow scalar or vector per-atom quantities calculated by a [fix](#) to be output. The ID in the keyword 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 keyword, then the scalar per-atom quantity 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 component of the M-length per-atom vector calculated by the fix.

The *v_name* keyword allows per-atom quantities calculated by a [variable](#) to be output. The name in the keyword 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:

Scaled coordinates cannot be written to dump files when the simulation box is triclinic (non-orthogonal). Note that this is the default for dump style *atom*; the [dump_modify command](#) must be used to change it. The exception is DCD files which store the tilt factors for subsequent visualization by programs like [VMD](#).

To write gzipped dump files, you must compile LAMMPS with the `-DLAMMPS_GZIP` option – see the [Making LAMMPS](#) section of the documentation.

The *bond* style is part of the "molecular" package. It is only enabled if LAMMPS was built with that package. See the [Making LAMMPS](#) section for more info.

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 lo-level XDR data format that XTC files are written with, which will result in a compile-time error when a lo-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 lo-level Makefile, e.g. `src/MAKE/Makefile.foo`. This compatibility mode has been tested successfully on Cray XT3 and IBM BlueGene/L machines and should also work on the Cray XT4, IBM BG/P, and Windows XP machines.

Related commands:

[dump_modify](#), [undump](#)

Default: none

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 = *format* or *scale* or *image* or *flush* or *unwrap* or *every* or *precision* or *region* or *thresh*

```
format arg = C-style format string for one line of output
scale arg = yes or no
image arg = yes or no
flush arg = yes or no
unwrap arg = yes or no
every arg = N
    N = dump every this many timesteps
precision arg = power-of-10 value from 10 to 1000000
region arg = region-ID or "none"
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
```

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
```

Description:

Modify the parameters of a previously defined dump command. Not all parameters are relevant to all dump styles.

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.

The *scale* and *image* keywords apply 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. A value of *no* means they are written in absolute distance units (e.g. Angstroms or sigma). 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 values can be printed in the dump file by using the appropriate atom attributes in the dump command itself.

The *flush* option 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 *unwrap* option 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.

The *every* option changes the dump frequency originally specified by the [dump](#) command to a new value which must be > 0 . The dump frequency cannot be changed for the dump *dcd* style.

The *precision* option 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* style. If specified, only atoms in the region will be written to the dump file. 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 *thresh* keyword only applies to the dump *custom* style. 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. The possible attributes that can be tested for are the same as those that can be specified in the [dump custom](#) command. 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.

Restrictions: none

Related commands:

[dump](#), [undump](#)

Default:

The option defaults are `format = %d` and `%g` for each integer or floating point value, `scale = yes`, `image = no`, `flush = yes` (except for the dump *xtc* style), `unwrap = no`, `every = whatever it was set to via the dump command`, `precision = 1000`, `region = none`, and `thresh = none`.

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 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 calculate a global scalar or vector quantity which can be accessed by various commands for output, including [equal- and atom-style variables](#), [thermo_style custom](#), and [fix ave/time](#).

Some fixes calculate a per-atom scalar or vector quantity which can be accessed by various commands for output, including [atom-style variables](#), [dump_style custom](#), and [fix ave/spatial](#).

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. timestep size. Extensive means the value scales with the number of atoms in the simulation, e.g. total force applied by the fix. [Thermodynamic output](#) will normalize extensive values depending on the "thermo_modify norm" setting. But if a fix value is accessed in another way, e.g. by a [variable](#), you may need to know whether it is an intensive or extensive value. See the doc page for individual fixes for further info.

See this [howto section](#) for a summary of various LAMMPS output options. See the doc pages for individual fixes for info on which ones calculate these quantities.

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.

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:

- [addforce](#) – add a force to each atom
- [aveforce](#) – add an averaged force to each atom
- [ave/atom](#) – compute per-atom time-averaged quantities
- [ave/spatial](#) – output per-atom quantities by layer
- [ave/time](#) – output time-averaged compute quantities
- [bond/break](#) – break bonds on the fly
- [bond/create](#) – create bonds on the fly
- [bond/swap](#) – Monte Carlo bond swapping
- [com](#) – compute a center-of-mass
- [coord/original](#) – store original coords of each atom
- [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
- [freeze](#) – freeze atoms in a granular simulation
- [gravity](#) – add gravity to atoms in a granular simulation
- [gyration](#) – compute radius of gyration
- [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
- [msd](#) – compute mean-squared displacement (i.e. diffusion coefficient)
- [momentum](#) – zero the linear and/or angular momentum of a group of atoms
- [nph](#) – constant NPH time integration via Nose/Hoover
- [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
- [rdf](#) – compute radial distribution functions
- [recenter](#) – constrain the center-of-mass position of a group of atoms
- [rigid](#) – constrain one or more clusters of atoms to move as a rigid body
- [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
- [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
- [viscosity](#) – Muller-Plathe momentum exchange for viscosity calculation
- [viscous](#) – viscous damping for granular simulations
- [wall/gran](#) – frictional wall(s) for granular simulations
- [wall/lj126](#) – Lennard-Jones 12–6 wall
- [wall/lj93](#) – Lennard-Jones 9–3 wall
- [wall/reflect](#) – reflecting wall(s)
- [wiggle](#) – oscillate walls and frozen atoms

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](#).

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 addforce command

Syntax:

```
fix ID group-ID addforce fx fy fz
```

- ID, group-ID are documented in [fix](#) command
- addforce = style name of this fix command
- fx,fy,fz = force component values (force units)

Examples:

```
fix kick flow addforce 1.0 0.0 0.0
```

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.

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 scalar and a 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 vector values calculated by this fix are "extensive", meaning they scale with the number of atoms in the simulation.

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 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 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 = calculate property every this many timesteps
- Nrepeat = # of times to repeat the Nevery calculation before averaging
- Nfreq = timestep frequency at which the average value is calculated
- one or more 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 = per-atom scalar value calculated by a compute with ID
c_ID[N] = Nth component of per-atom vector calculated by a compute with ID
f_ID = per-atom scalar value calculated by a fix with ID
f_ID[N] = Nth component of per-atom vector calculated by a fix with ID
v_name = per-atom value 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:

Calculate one or more instantaneous per-atom quantities every few timesteps, and average them 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.

Each listed value is averaged independently. The group specified with the command means only atoms within the group have their averages computed. Atoms not in the group have their result(s) set to 0.0.

Each listed 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 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 *Nevery*, *Nrepeat*, and *Nfreq* arguments specify on what timesteps the values will be generated in order to contribute to the average. The final averaged quantities are generated every *Nfreq* timesteps. 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.

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 scalar calculated by the compute is used. If a bracketed term is appended, the Nth vector per-atom value 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 scalar calculated by the fix is used. If a bracketed term is appended, the Nth vector per-atom value 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 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 scalar or vector which can be accessed by various [output commands](#). A scalar is produced if only a single quantity is averaged by this fix. If two or more quantities are averaged, then a vector 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/spatial](#), [dump custom](#)

Default: none

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 = calculate property every this many timesteps
- Nrepeat = # of times to repeat the Nevery calculation before averaging
- Nfreq = timestep frequency at which the average value is computed
- dim = *x* or *y* or *z*
- origin = *lower* or *center* or *upper* or coordinate value (distance units)
- delta = thickness of spatial layers in dim (distance units)
- one or more values can be listed
- value = *x*, *y*, *z*, *vx*, *vy*, *vz*, *fx*, *fy*, *fz*, density/mass, density/number, *c_ID*, *c_ID[N]*, *f_ID*, *f_ID[N]*, *v_name*

```
x,y,z,vx,vy,vz,fx,fy,fz = atom attribute (velocity, force component)
density/number, density/mass = number or mass density
c_ID = per-atom scalar value calculated by a compute with ID
c_ID[N] = Nth component of per-atom vector calculated by a compute with ID
f_ID = per-atom scalar value calculated by a fix with ID
f_ID[N] = Nth component of per-atom vector calculated by a fix with ID
v_name = per-atom value 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*

```
units arg = box or lattice or reduced
norm arg = all or sample
file arg = filename
filename = file to write results to
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
```

Examples:

```
fix 1 all ave/spatial 10000 1 10000 z lower 0.02 c_myCentro units reduced
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 y 0.0 2.5 density/mass ave running
```

Description:

Calculate one or more instantaneous per-atom quantities every few timesteps, average them by layer in a chosen dimension, and average the layer values over a longer timescale. The resulting averages can be used by other [output commands](#) such as [thermo_style custom](#), and can also be written to a file.

Each listed value is averaged independently. The group specified with the command means only atoms within the group contribute to the layer 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 *Nevery*, *Nrepeat*, and *Nfreq* arguments specify on what timesteps the layer values will be generated in order to contribute to the average. The final averaged quantities are generated every *Nfreq* timesteps. 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 layer, where the layers are in a particular *dim* and have a thickness given by *delta*. Every *Nfreq* steps, when an averaging is being performed and the per-atom property is calculated for the first time, the number of layers and the layer boundaries are computed. Thus if the simulation box changes size during a simulation, the number of layers and their boundaries may also change. Layers are defined relative to a specified *origin*, which may be the lower/upper edge of the box (in *dim*) or its center point, or a specified coordinate value. Starting at the origin, sufficient layers are created in both directions to completely cover the box. On subsequent timesteps every atom is mapped to one of the layers. Atoms beyond the lowermost/uppermost layer are counted in the first/last layer.

For orthogonal simulation boxes, the layers are "slices" aligned with the xyz coordinate axes. For non-orthogonal (triclinic) simulation boxes, the layers are "tilted slices" 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. As described there, a tilted simulation box has edge vectors a,b,c. In that nomenclature, layers in the x dimension have faces with normals in the "b" cross "c" direction. Layers in y have faces normal to the "a" cross "c" direction. And layers in z have faces normal to the "a" cross "b" direction. Note that in order to define the thickness and position of these tilted layers in an unambiguous fashion, the *units* option must be set to *reduced* when using a non-orthogonal simulation box, as discussed below.

The atom attribute values (x,y,z,vx,vy,vz,fx,fy,fz) are self-explanatory.

The *density/number* value means the number density is computed in each layer, i.e. a weighting of 1 for each atom. The *density/mass* value means the mass density is computed in each layer, i.e. each atom is weighted by its mass. The resulting density is normalized by the volume of the layer so that units of number/volume or mass/volume are output.

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 scalar calculated by the compute is used. If a bracketed term is appended, the Nth vector per-atom value 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 scalar calculated by the fix is used. If a bracketed term is appended, the Nth vector per-atom value 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, 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 layer thickness *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 layers span the box in any dimension.

Consider a non-orthogonal box, with layers 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 layer quantity is summed over all atoms in all *Nrepeat* samples, as is the count of atoms in the layer. The printed value for the layer is Total-quantity / Total-count. In other words it is an average over the entire *Nfreq* timescale.

For a *sample* setting, the layer 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 layer is the average of the *Nrepeat* "average sample values", In other words it is an average of an average.

The *file* keyword allows a filename to be specified. Every *Nfreq* timesteps, layer info will be written to a text file in the following format. A line with the timestep and number of layers is written. Then one line per layer is written, containing the layer ID (1-N), the coordinate of the center of the layer, the number of atoms in the layer, 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 *ave* keyword determines how the layer values produced every *Nfreq* steps are averaged with layer 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 layer 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 layer values produced on timesteps that are multiples of *Nfreq* are summed and averaged in a cumulative sense before being output. Each output layer value is thus the average of the layer value produced on that timestep with all preceding values for the same layer. 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 layer 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 layer 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 layer values on steps 8000,9000,10000. Outputs on early steps will average over less than *M* values if they are not available.

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 vector of quantities 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 vector is of length $N = n_{\text{layers}} * n_{\text{values}}$ where *nlayers* is the number of layers and *nvalues* is the number of values per layer that the fix is averaging. When accessed by another output command, a single index *M* is specified which is mapped into a layer *I* as $I = M / n_{\text{values}} + 1$ and into value *J* as $J = M \% n_{\text{values}} + 1$. If *I* exceeds the current number of layers then a 0.0 is returned by the fix instead of an error, since the number of layers can vary as a simulation runs, depending on the simulation box size. The vector values calculated by this fix are "intensive", meaning they are independent of the number of atoms in the simulation, since they are already normalized by the count of atoms in each layer.

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 layers 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/time](#)

Default:

The option defaults are *units* = *lattice*, *norm* = *all*, no file output, and *ave* = *one*.

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 = calculate properties every this many timesteps
- Nrepeat = # of times to repeat the Nevery calculation before averaging
- Nfreq = timestep frequency at which averages are computed
- one or more values can be listed
- value = c_ID, c_ID[N], f_ID, f_ID[N], v_name

```
c_ID = global scalar value calculated by a compute with ID
c_ID[N] = Nth component of global vector calculated by a compute with ID
f_ID = global scalar value calculated by a fix with ID
f_ID[N] = Nth 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 = *file* or *ave*

```
file arg = filename
    filename = name of file to output time averages to
ave args = one or running or window M
    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
```

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
fix 1 all ave/time 1 100 1000 f_indent f_indent[1] file temp.indent
```

Description:

Calculate one or more instantaneous global quantities 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. If no averaging is done, this command is a convenient way to simply write one or more desired quantities to a separate file.

Each listed value is averaged independently. If written to a file, then over time, one column of numbers is produced for each value. The group specified with the command is ignored, since calculations are performed by computes and fixes which store their own "group" definition,

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 quantity. If you wish to spatial- or time-average per-atom quantities from a compute, fix, or variable, then see the [fix ave/spatial](#) or [fix ave/atom](#) 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 *Nevery*, *Nrepeat*, and *Nfreq* arguments specify on what timesteps the values will be generated in order to contribute to the average. The final averaged quantities are generated every *Nfreq* timesteps. 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 no bracketed term is appended, the global scalar calculated by the compute is used. If a bracketed term is appended, the Nth vector value 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 no bracketed term is appended, the global scalar calculated by the fix is used. If a bracketed term is appended, the Nth vector value 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. 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 generating quantities to time average.

Additional optional keywords also affect the operation of this fix.

The *file* keyword allows a filename to be specified. Each timestamp, one quantity is written to the file for each value specified in the fix ave/time command. The file is in a self-explanatory text format.

The *ave* keyword determines how the scalar and/or vector 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 cumulative 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 this value is 0.0, so setting *start* to a larger value can avoid including a 0.0 in a running or windowed 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.

This fix produces a global scalar or vector which can be accessed by various [output commands](#). A scalar is produced if only a single quantity is averaged by this fix. If two or more quantities are averaged, then a vector of values is produced. The global values can only be accessed on timesteps that are multiples of *Nfreq* since that is when averaging is performed. Each value (scalar or vector component) calculated by this fix may be either "intensive" or "extensive". Intensive means the value is independent of the number of atoms in the simulation. Extensive means the value scales with the number of atoms in the simulation. 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 whatever the variable calculates.

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](#), [variable](#), [fix ave/spatial](#), [fix ave/atom](#)

Default: none

The option defaults no file output, ave = one, and start = 0.

fix aveforce command

Syntax:

```
fix ID group-ID aveforce fx fy fz
```

- ID, group-ID are documented in [fix](#) command
- aveforce = style name of this fix command
- fx,fy,fz = force component values (force units)

Examples:

```
fix pressdown topwall aveforce 0.0 -1.0 0.0  
fix 2 bottomwall aveforce NULL -1.0 0.0
```

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.

If any of the arguments is specified as NULL then the forces in that dimension are 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.

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 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", meaning they scale with the number of atoms in the simulation.

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 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 as its sole partner (due to $R_{jk} > R_{ij}$), 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 bond](#) 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 equilibrium 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 vector of length 2, which can be accessed by various [output commands](#). The vector values calculated by this fix are "extensive", meaning they scale with the number of atoms in the simulation.

These are the 2 quantities:

- (1) # of bonds broken on the most recent breakage timestep
- (2) cumulative # 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:

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 bond](#), [special_bonds](#)

Default:

The option defaults are $\text{prob} = 1.0$.

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 as its sole partner (due to $R_{jk} < R_{ij}$), 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 \leq 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 bond](#) 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 equilibrium 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 vector of length 2, which can be accessed by various [output commands](#). The vector values calculated by this fix are "extensive", meaning they scale with the number of atoms in the simulation.

These are the 2 quantities:

- (1) # of bonds created on the most recent creation timestep
- (2) cumulative # 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:

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 dipoles, 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 bond](#), [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)

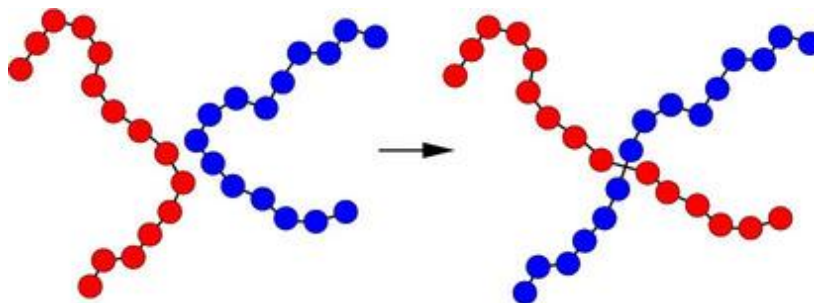
Examples:

```
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](#) bead-spring polymer chains where each polymer is a linear chain of monomers, but LAMMPS does not enforce this requirement.

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 Δ 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 feature you must setup the molecule IDs for your polymer chains in a certain way, typically in the data file, read by the [read_data](#) command. 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 monomer 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 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.

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 2–vector of output, which can be accessed by various [output commands](#). The first component of the vector is the cumulative number of swaps performed by all processors. The second component of the vector is the cumulative 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", meaning they are independent of the number of atoms in the simulation.

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 settings of the "special_bond" command must be 0,1,1 in order to use this fix, which is typical of FENE bead–spring chains. I.e. 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 com command

Syntax:

```
fix ID group-ID com N file
```

- ID, group-ID are documented in [fix](#) command
- com = style name of this fix command
- N = compute center-of-mass every this many timesteps
- file = filename to write center-of-mass info to

Examples:

```
fix 1 all com 100 com.out
```

Description:

Compute the center-of-mass of the group of atoms every N steps, including all effects due to atoms passing thru periodic boundaries. Write the results to the specified 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. 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: none

Related commands: none

Default: none

fix coord/original command

Syntax:

```
fix ID group-ID coord/original
```

- ID, group-ID are documented in [fix](#) command
- coord/original = style name of this fix command

Examples:

```
fix 1 all coord/original
```

Description:

Store the original coordinates of atoms in the group at the time the fix command is issued. This is useful for computing a displacement of the atoms at later times, via the [compute displace/atom](#) command. Or the original coordinates can be accessed by other [output commands](#) that use per-atom quantities such as the [dump custom](#) command.

IMPORTANT NOTE: To enable displacements to be calculated which include the effects of atoms passing through periodic boundaries, the original coordinates 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.

IMPORTANT NOTE: If an atom is part of a rigid body (see the [fix rigid](#) command), its periodic image flags are altered, and its original coordinates may not be what you expect. See the [fix rigid](#) command for details.

Restart, fix_modify, output, run start/stop, minimize info:

This fix writes the original coordinates of the atoms 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.

This fix produces a per-atom vector which can be accessed by various [output commands](#). The length of the vector for each atom is 3, and the components store the x,y,z coords of 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 msd](#)

Default: none

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
  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
xy, xz, yz args = style value
style = final or delta or vel or erate or trate
  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)
```

- 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 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" 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. Unspecified periodic dimensions 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 $\text{rate} = V/L_0$ and L_0 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 $\text{delta}/\text{length}_0$, where length_0 is the original box length and delta is the change relative to the original length. Thus if the *erate* R is 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 10, not 1024 as it would with *trate*.

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 Δ/length_0 , where length_0 is the original box length and Δ is the change relative to the original length. Thus if the *trate* R is 0.1 and time units are picoseconds, this means the box length will increase by 10% of its current length every picosecond. I.e. strain after 1 psec = 0.1, strain after 2 psec = 0.21, etc. $R = 1$ or 2 means the box length will double or triple every picosecond. $R = -0.01$ 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 $R = 1$ for 10 picoseconds does not expand the box length by a factor of 10 as it would with *erate*, but by a factor of 1024 since it doubles 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 every picosecond, you could set "x *trate* M", "y *trate* M", "z *trate* M", with $M = \text{pow}(2,1/3) - 1 = 1.26$, since if each box dimension grows by 26%, the box volume doubles.

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.

For the *scale*, *vel*, *erate*, *trate*, 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.

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 $\text{rate} = V/L_0$ and L_0 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. Thus if the *erate* R is 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, $0.1 \cdot y_{\text{box}}$ at 1 psec, $0.2 \cdot y_{\text{box}}$ at 2 psec, etc, where y_{box} 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. Thus if the *trate* R is 0.1 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. $R = 1$ or 2 means the tilt factor will double or triple every picosecond. $R = -0.01$ means the tilt factor will shrink by 1% every picosecond. Note that the change is continuous, so running with $R = 1$ 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. Also 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.

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 $x_{\text{lo}} = 2$ and $x_{\text{hi}} = 12$, 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 $-(x_{\text{hi}} - x_{\text{lo}})/2$ and $+(y_{\text{hi}} - y_{\text{lo}})/2$. Note that this is not a limitation, since if the maximum tilt factor is 5 (as in this example), then configurations with tilt = ..., $-15, -5, 5, 15, 25, \dots$ 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 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 *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.

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 provide 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 scalar or vector 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 units)
    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 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 deposited, 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 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: none

Related commands:

`fix_pour`, `region`

Default:

The option defaults are $\text{delta} = 0.0$, $\text{near} = 0.0$, $\text{attempt} = 10$, $\text{rate} = 0.0$, $\text{vx} = 0.0\ 0.0$, $\text{vy} = 0.0\ 0.0$, $\text{vz} = 0.0\ 0.0$, and $\text{units} = \text{lattice}$.

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 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", meaning they scale with the number of atoms in the simulation.

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 INF) (time units)
- Tmax = maximum dt allowed (can be INF) (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, 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 way.

- compute Vmax of any atom in group
- compute Amax of any atom in group
- $dt1 = Xmax/Vmax$
- $dt2 = \sqrt{2 \cdot Xmax/Amax}$
- new dt = MIN(dt1,dt2)
- if $dt < Tmin$, $dt = Tmin$
- if $dt > Tmax$, $dt = Tmax$

Vmax is the maximum velocity; Amax is the maximum acceleration = force/mass. Note that Tmin or Tmax can be specified as INF, in which case one or both of the last 2 checks will not be performed.

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.

The current timestep size is stored as a scalar quantity by this fix. The cumulative simulation time (in time units) is stored as the first element of a vector. Both these quantities can be accessed by various [output commands](#). The scalar and vector values calculated by this fix are "intensive", meaning they are independent of the number of atoms in the simulation.

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)

Examples:

```
fix kick external-field efield 1.0 0.0 0.0
```

Description:

Add a force $F = qE$ to each charged atom in the group due to an external electric field being applied to the system.

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: none

Related commands:

[fix addforce](#)

Default: none

fix enforce2d 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.

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.

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 freeze 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.

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 "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 parts of the code (pair potentials, thermodynamics, etc) 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 granular](#)

Default: none

fix gravity 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 ϕ, θ at the time the fix is specified. For 2d systems, ϕ and ϕ_{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.

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: none

Related commands:

[atom_style granular](#), [fix addforce](#)

Default: none

fix gyration command

Syntax:

```
fix ID group-ID gyration N file
```

- ID, group-ID are documented in [fix](#) command
- gyration = style name of this fix command
- N = compute radius-of-gyration every this many timesteps
- file = filename to write gyration info to

Examples:

```
fix 1 all gyration 100 molecule.out
```

Description:

Compute the radius-of-gyration of the group of atoms every N steps, including all effects due to atoms passing thru periodic boundaries. Write the results to the specified file.

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 and Rcm is the center-of-mass position of the group.

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: none

Related commands: none

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)

Examples:

```
fix 3 qin heat 1 1.0  
fix 4 qout heat 1 -1.0
```

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 to one group of atoms (hot reservoir) and subtracting heat from another (cold reservoir). E.g. a simulation sampling from the McDLT ensemble. Note that the fix is applied to a group of atoms, not a geometric region, so that the same set of atoms is 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. Since eflux is in units of energy/time, this means a larger value of N will add/subtract a larger amount of energy each timestep the fix is invoked. If heat is subtracted from the system too aggressively so that the group's kinetic energy goes 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. 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: none

Related commands:

[compute temp](#), [compute temp/region](#)

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 *vel* or *rstart* 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)
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)
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)
  side = lo or hi
vel args = vx vy vz
  vx,vy,vz = velocity of center of indenter (velocity units)
rstart value = R0
  R0 = sphere or cylinder radius at start of run (distance units)
  R is value at end of run, so indenter expands/contracts over time
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 vel 0.0 0.0 -1.0
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.

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.

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*.

If the *vel* keyword is specified, the center (or axis or position) of the spherical (or cylindrical or planar) indenter will move during the simulation, based on its initial position (x,y,z) and the specified (vx,vy,vz). Note that if you do multiple runs, the initial position of the indenter (x,y,z) does not change, so it will continue to move at the specified velocity.

If the *rstart* keyword is specified, then the radius of the indenter is a time-dependent quantity. R_0 is the value assigned at the start of the run; R is the value at the end. At intermediate times, the radius is linearly interpolated between these two values. This option can be used, for example, to grow/shrink a void within the simulation box.

The *units* keyword determines the meaning of the distance units used to define the indenter. 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 not only the indenter's physical geometry, but also its velocity and force constant since they are defined in terms of distance as well.

IMPORTANT NOTE: For spherical and cylindrical indenters, you should insure the indenter's extent does not overlap a periodic boundary, either for a fixed indenter, or one that moves. No check for such overlaps is performed by the code.

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 scalar energy and a 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", meaning they scale with the number of atoms in the simulation.

This fix can adjust the indenter position and radius over multiple runs, using the *start* and *stop* keywords of the [run](#) command. See the [run](#) command for details of how to do this.

The forces due to this fix are imposed during an energy minimization, invoked by the [minimize](#) command. The *rstart* keyword does not change the indenter radius during an energy minimization; the indenter always has a radius of its final value R in that case.

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 $\text{vel} = 0,0,0$ and $\text{units} = \text{lattice}$.

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 = scale
scale values = type ratio
type = atom type (1-N)
ratio = factor by which to scale the damping coefficient
```

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 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:

$$F = F_c + F_f + F_r$$

$$F_f = - (m / \text{damp}) v$$

F_r is proportional to $\sqrt{K_b T m / (dt \text{ damp})}$

F_c is the conservative force computed via the usual inter-particle interactions ([pair_style](#), [bond_style](#), etc).

The F_f and F_r terms are added by this fix.

F_f 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.

F_r 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 $\sqrt{K_b T m / (dt \text{ damp})}$, where K_b 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 thermostating takes place; see the description below.

IMPORTANT NOTE: Unlike the [fix nvt](#) command which performs Nose/Hoover thermostating AND time integration, this fix does NOT perform time integration. It only modifies forces to effect thermostating. 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 thermostating.

The desired temperature at each timestep is a ramped value during the run from T_{start} to T_{stop} .

Like other fixes that perform thermostating, 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, thermostating 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 *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 $F\tau$, unlike *fix viscous*. Thus the mass and size of different atom types should be accounted for in the choice of ratio values.

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 thermostating procedure, as described above. For consistency, the group used by this fix and by the compute should be the same.

No global scalar or vector or per-atom quantities are stored by this fix for access by various [output commands](#).

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](#)

Default:

The option defaults are axes = 1 1 1, scale = 1.0 for all types, no region, and weight = 1.0.

(Dunweg) Dunweg and Paul, Int J of Modern Physics C, 2, 817–27 (1991).

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 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.

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](#) thermostating).

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 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: none

Related commands:

[fix recenter](#), [velocity](#)

Default: none

fix msd command

Syntax:

```
fix ID group-ID msd N file keyword value ...
```

- ID, group-ID are documented in [fix](#) command
- msd = style name of this fix command
- N = compute mean-squared displacement every this many timesteps
- file = filename to write mean-squared displacement info to
- zero or more keyword/value pairs may be appended
- keyword = *com*

com value = yes or no

Examples:

```
fix 1 all msd 100 diff.out  
fix 1 all msd 1000 diff.out com yes
```

Description:

Compute the mean-squared displacement (MSD) of the group of atoms every N steps, including all effects due to atoms passing thru periodic boundaries. The MSD is the square of the displacement of each atom, averaged over all the atoms in the group. The displacement of an atom is from its original position at the time the fix command was issued. The slope of the mean-squared displacement versus time is proportional to the diffusion coefficient of the diffusing atoms.

The MSD values versus time are written to the specified file.

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 MSD will not reflect its true displacement. 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.

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 displacement of each atom is calculated.

Restart, fix_modify, output, run start/stop, minimize info:

This fix writes the original coordinates of diffusing atoms to [binary restart files](#), so that the mean-squared displacement will be accurate 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. 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: none

Related commands:

[fix coord/original](#)

Default:

The option default is com = no.

fix nph command

Syntax:

```
fix ID group-ID nph p-style args keyword value ...
```

- ID, group-ID are documented in [fix](#) command
- nph = style name of this fix command
- p-style = *xyz* or *xy* or *yz* or *xz* or *aniso*

```
xyz args = Pstart Pstop Pdamp
          Pstart,Pstop = desired pressure at start/end of run (pressure units)
          Pdamp = pressure damping parameter (time units)
xy or yz or xz args = Px0 Px1 Py0 Py1 Pz0 Pz1 Pdamp
                    Px0,Px1,Py0,Py1,Pz0,Pz1 = desired pressure in x,y,z at
                    start/end (0/1) of run (pressure units)
                    Pdamp = pressure damping parameter (time units)
aniso args = Px0 Px1 Py0 Py1 Pz0 Pz1 Pdamp
            Px0,Px1,Py0,Py1,Pz0,Pz1 = desired pressure in x,y,z at
            start/end (0/1) of run (pressure units)
            Pdamp = pressure damping parameter (time units)
```

- zero or more keyword/value pairs may be appended
- keyword = *drag* or *dilate*

```
drag value = drag factor added to barostat (0.0 = no drag)
dilate value = all or partial
```

Examples:

```
fix 1 all nph xyz 0.0 0.0 1000.0
fix 2 all nph xz 5.0 5.0 NULL NULL 5.0 5.0 1000.0 drag 1.0
fix 2 all nph aniso 0.0 0.0 0.0 0.0 NULL NULL 1000.0
```

Description:

Perform constant NPH integration to update positions and velocities each timestep for atoms in the group using a Nose/Hoover pressure barostat ([Hoover](#)), implemented as described in ([Melchionna](#)). P is pressure. This creates a system trajectory consistent with the isobaric ensemble. Unlike [fix npt](#), temperature will not be controlled if no other fix is used. Temperature can be controlled independently by using a thermostating fix such as [fix langevin](#) or [fix temp/rescale](#).

The atoms 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 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 press/berendsen](#) command which performs barostatting but NO time integration, this fix performs barostatting AND time integration. Thus you should not use any other time integration fix, such as [fix nve](#) or [fix nvt](#) on atoms to which this fix is applied. Use [fix npt](#) instead of this fix, if

you want to control both temperature and pressure via Nose/Hoover.

See [this howto section](#) of the manual for a discussion of different ways to compute temperature and perform thermostating and barostatting.

The pressure can be controlled in one of several styles, as specified by the *p-style* argument. In each case, the desired pressure at each timestep is a ramped value during the run from the starting value to the end value.

Style *xyz* means couple all dimensions together when pressure is computed (isotropic pressure), and dilate/contract the dimensions together.

Styles *xy* or *yz* or *xz* means that the 2 specified dimensions are coupled together, both for pressure computation and for dilation/contraction. The 3rd dimension dilates/contracts independently, using its pressure component as the driving force. These styles cannot be used for a 2d simulation.

For style *aniso*, all dimensions dilate/contract independently using their individual pressure components as the driving forces.

For any of the styles except *xyz*, any of the independent pressure components (e.g. *z* in *xy*, or any dimension in *aniso*) can have their target pressures (both start and stop values) specified as NULL. This means that no pressure control is applied to that dimension so that the box dimension remains unchanged. For a 2d simulation the *z* pressure components must be specified as NULL when using style *aniso*.

For styles *xy* and *yz* and *xz*, the starting and stopping pressures must be the same for the two coupled dimensions and cannot be specified as NULL.

In some cases (e.g. for solids) the pressure (volume) and/or temperature of the system can oscillate undesirably when a Nose/Hoover barostat 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 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.

For all pressure styles, the simulation box stays rectangular in shape. Parinello–Rahman boundary condition for tilted boxes (triclinic symmetry) are supported by other LAMMPS commands (see [this section](#) of the manual), but not yet by this command.

For all styles, 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).

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:

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 thermostating 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](#).

The potential energy change due to this fix is stored as a scalar quantity, which can be accessed by various [output commands](#). The scalar value calculated by this fix is "extensive", meaning it scales with the number of atoms in the simulation.

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. A dimension whose target pressures are specified as NULL can be non-periodic or periodic.

Related commands:

[fix nve](#), [fix npt](#), [fix_modify](#)

Default:

The keyword defaults are *drag* = 0.0 and *dilate* = all.

(Hoover) Hoover, Phys Rev A, 34, 2499 (1986).

(Melchionna) Melchionna, Ciccotti, Holian, Molecular Physics, 78, 533–44 (1993).

fix npt command

Syntax:

```
fix ID group-ID npt Tstart Tstop Tdamp p-style args keyword value ...
```

- ID, group-ID are documented in [fix](#) command
- npt = style name of this fix command
- Tstart,Tstop = desired temperature at start/end of run
- Tdamp = temperature damping parameter (time units)
- p-style = *xyz* or *xy* or *yz* or *xz* or *aniso*

```
xyz args = Pstart Pstop Pdamp
Pstart,Pstop = desired pressure at start/end of run (pressure units)
Pdamp = pressure damping parameter (time units)
xy or yz or xz or aniso args = Px_start Px_stop Py_start Py_stop Pz_start Pz_stop Pdamp
Px_start,Px_stop,... = desired pressure in x,y,z at start/end of run (pressure units)
Pdamp = pressure damping parameter (time units)
```

- zero or more keyword/value pairs may be appended
- keyword = *drag* or *dilate*

```
drag value = drag factor added to barostat/thermostat (0.0 = no drag)
dilate value = all or partial
```

Examples:

```
fix 1 all npt 300.0 300.0 100.0 xyz 0.0 0.0 1000.0
fix 2 all npt 300.0 300.0 100.0 xz 5.0 5.0 NULL NULL 5.0 5.0 1000.0
fix 2 all npt 300.0 300.0 100.0 xz 5.0 5.0 NULL NULL 5.0 5.0 1000.0 drag 0.2
fix 2 water npt 300.0 300.0 100.0 aniso 0.0 0.0 0.0 0.0 NULL NULL 1000.0 dilate partial
```

Description:

Perform constant NPT integration to update positions and velocities each timestep for atoms in the group using a Nose/Hoover temperature thermostat ([Hoover1](#)) and Nose/Hoover pressure barostat ([Hoover2](#)), implemented as described in ([Melchionna](#)). P is pressure; T is temperature. This creates a system trajectory consistent with the isothermal–isobaric ensemble.

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 from them before thermostating 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).

The atoms 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 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 temp/berendsen](#) command which performs thermostating but NO time integration, this fix performs thermostating/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, this fix 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 thermostating and barostatting.

The pressure can be controlled in one of several styles, as specified by the *p-style* argument. In each case, the desired pressure at each timestep is a ramped value during the run from the starting value to the end value.

Style *xyz* means couple all dimensions together when pressure is computed (isotropic pressure), and dilate/contract the dimensions together.

Styles *xy* or *yz* or *xz* means that the 2 specified dimensions are coupled together, both for pressure computation and for dilation/contraction. The 3rd dimension dilates/contracts independently, using its pressure component as the driving force. These styles cannot be used for a 2d simulation.

For style *aniso*, all dimensions dilate/contract independently using their individual pressure components as the driving forces.

For any of the styles except *xyz*, any of the independent pressure components (e.g. *z* in *xy*, or any dimension in *aniso*) can have their target pressures (both start and stop values) specified as NULL. This means that no pressure control is applied to that dimension so that the box dimension remains unchanged. For a 2d simulation the *z* pressure components must be specified as NULL when using style *aniso*.

For styles *xy* and *yz* and *xz*, the starting and stopping pressures must be the same for the two coupled dimensions and cannot be specified as NULL.

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.

For all pressure styles, the simulation box stays rectangular in shape. Parinello–Rahman boundary condition for tilted boxes (triclinic symmetry) are supported by other LAMMPS commands (see [this section](#) of the manual), but not yet by this command.

For all styles, the *Pdamp* parameter operates like the *Tdamp* parameter, determining 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).

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.

Like other fixes that perform thermostating, 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, thermostating 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 thermostating 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 this fix to add the energy change induced by Nose/Hoover thermostating and barostatting to the system's potential energy as part of [thermodynamic output](#).

The potential energy change due to this fix is stored as a scalar quantity, which can be accessed by various [output commands](#). The scalar value calculated by this fix is "extensive", meaning it scales with the number of atoms in the simulation.

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:

Any dimension being adjusted by this fix must be periodic. A dimension whose target pressures are specified as NULL can be non-periodic or periodic.

The final Tstop cannot be 0.0 since it would make the target $T = 0.0$ at some timestep during the simulation which is not allowed in the Nose/Hoover formulation.

Related commands:

[fix nve](#), [fix nvt](#), [fix nph](#), [fix_modify](#)

Default:

The keyword defaults are drag = 0.0 and dilate = all.

(Hoover1) Hoover, Phys Rev A, 31, 1695 (1985).

(Hoover2) Hoover, Phys Rev A, 34, 2499 (1986).

(Melchionna) Melchionna, Ciccotti, Holian, Molecular Physics, 78, 533–44 (1993).

fix npt/asphere command

Syntax:

```
fix ID group-ID npt/asphere Tstart Tstop Tdamp p-style args keyword value ...
```

- ID, group-ID are documented in [fix](#) command
- npt/asphere = style name of this fix command
- Tstart,Tstop = desired temperature at start/end of run
- Tdamp = temperature damping parameter (time units)
- p-style = *xyz* or *xy* or *yz* or *xz* or *aniso*

```
xyz args = Pstart Pstop Pdamp
Pstart,Pstop = desired pressure at start/end of run (pressure units)
Pdamp = pressure damping parameter (time units)
xy or yz or xz or aniso args = Px_start Px_stop Py_start Py_stop Pz_start Pz_stop Pdamp
Px_start,Px_stop,... = desired pressure in x,y,z at start/end of run (pressure units)
Pdamp = pressure damping parameter (time units)
```

- zero or more keyword/value pairs may be appended
- keyword = *drag* or *dilate*

```
drag value = drag factor added to barostat/thermostat (0.0 = no drag)
dilate value = all or partial
```

Examples:

```
fix 1 all npt/asphere 300.0 300.0 100.0 xyz 0.0 0.0 1000.0
fix 2 all npt/asphere 300.0 300.0 100.0 xz 5.0 5.0 NULL NULL 5.0 5.0 1000.0
fix 2 all npt/asphere 300.0 300.0 100.0 xz 5.0 5.0 NULL NULL 5.0 5.0 1000.0 drag 0.2
fix 2 water npt/asphere 300.0 300.0 100.0 aniso 0.0 0.0 0.0 0.0 NULL NULL 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.

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.

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 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.

The pressure can be controlled in one of several styles, as specified by the *p-style* argument. In each case, the desired pressure at each timestep is a ramped value during the run from the starting value to the end value.

Style *xyz* means couple all dimensions together when pressure is computed (isotropic pressure), and dilate/contract the dimensions together.

Styles *xy* or *yz* or *xz* means that the 2 specified dimensions are coupled together, both for pressure computation and for dilation/contraction. The 3rd dimension dilates/contracts independently, using its pressure component as the driving force. These styles cannot be used for a 2d simulation.

For style *aniso*, all dimensions dilate/contract independently using their individual pressure components as the driving forces.

For any of the styles except *xyz*, any of the independent pressure components (e.g. *z* in *xy*, or any dimension in *aniso*) can have their target pressures (both start and stop values) specified as NULL. This means that no pressure control is applied to that dimension so that the box dimension remains unchanged. For a 2d simulation the *z* pressure components must be specified as NULL when using style *aniso*.

For styles *xy* and *yz* and *xz*, the starting and stopping pressures must be the same for the two coupled dimensions and cannot be specified as NULL.

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.

For all pressure styles, the simulation box stays rectangular in shape. Parinello–Rahman boundary conditions (tilted box) are not yet implemented in LAMMPS.

For all styles, the *Pdamp* parameter operates like the *Tdamp* parameter, determining 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).

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 group-ID temp/asphere
compute fix-ID_press group-ID 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 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.

Like other fixes that perform thermostating, 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, thermostating 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 thermostating 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 thermostating and barostatting to the system's potential energy as part of [thermodynamic output](#).

The potential energy change due to this fix is stored as a scalar quantity, which can be accessed by various [output commands](#).

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 particles be represented as extended ellipsoids and not point particles. This means they will have an angular momentum and a shape which is determined by the [shape](#) command.

Any dimension being adjusted by this fix must be periodic. A dimension whose target pressures are specified as NULL can be non-periodic or periodic.

The final Tstop cannot be 0.0 since it would make the target $T = 0.0$ at some timestep during the simulation which is not allowed in the Nose/Hoover formulation.

Related commands:

[fix npt](#), [fix nve_asphere](#), [fix npt_asphere](#), [fix_modify](#)

Default:

The keyword defaults are `drag = 0.0` and `dilate = all`.

fix npt/sphere command

Syntax:

```
fix ID group-ID npt/sphere Tstart Tstop Tdamp p-style args keyword value ...
```

- ID, group-ID are documented in [fix](#) command
- npt/sphere = style name of this fix command
- Tstart,Tstop = desired temperature at start/end of run
- Tdamp = temperature damping parameter (time units)
- p-style = *xyz* or *xy* or *yz* or *xz* or *aniso*

```
xyz args = Pstart Pstop Pdamp
Pstart,Pstop = desired pressure at start/end of run (pressure units)
Pdamp = pressure damping parameter (time units)
xy or yz or xz or aniso args = Px_start Px_stop Py_start Py_stop Pz_start Pz_stop Pdamp
Px_start,Px_stop,... = desired pressure in x,y,z at start/end of run (pressure units)
Pdamp = pressure damping parameter (time units)
```

- zero or more keyword/value pairs may be appended
- keyword = *drag* or *dilate*

```
drag value = drag factor added to barostat/thermostat (0.0 = no drag)
dilate value = all or partial
```

Examples:

```
fix 1 all npt/sphere 300.0 300.0 100.0 xyz 0.0 0.0 1000.0
fix 2 all npt/sphere 300.0 300.0 100.0 xz 5.0 5.0 NULL NULL 5.0 5.0 1000.0
fix 2 all npt/sphere 300.0 300.0 100.0 xz 5.0 5.0 NULL NULL 5.0 5.0 1000.0 drag 0.2
fix 2 water npt/sphere 300.0 300.0 100.0 aniso 0.0 0.0 0.0 0.0 NULL NULL 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 thermostating 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).

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.

The pressure can be controlled in one of several styles, as specified by the *p-style* argument. In each case, the desired pressure at each timestep is a ramped value during the run from the starting value to the end value.

Style *xyz* means couple all dimensions together when pressure is computed (isotropic pressure), and dilate/contract the dimensions together.

Styles *xy* or *yz* or *xz* means that the 2 specified dimensions are coupled together, both for pressure computation and for dilation/contraction. The 3rd dimension dilates/contracts independently, using its pressure component as the driving force. These styles cannot be used for a 2d simulation.

For style *aniso*, all dimensions dilate/contract independently using their individual pressure components as the driving forces.

For any of the styles except *xyz*, any of the independent pressure components (e.g. *z* in *xy*, or any dimension in *aniso*) can have their target pressures (both start and stop values) specified as NULL. This means that no pressure control is applied to that dimension so that the box dimension remains unchanged. For a 2d simulation the *z* pressure components must be specified as NULL when using style *aniso*.

For styles *xy* and *yz* and *xz*, the starting and stopping pressures must be the same for the two coupled dimensions and cannot be specified as NULL.

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.

For all pressure styles, the simulation box stays rectangular in shape. Parinello–Rahman boundary conditions (tilted box) are not yet implemented in LAMMPS.

For all styles, the *Pdamp* parameter operates like the *Tdamp* parameter, determining 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).

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 group-ID temp/sphere
compute fix-ID_press group-ID 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 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.

Like other fixes that perform thermostating, 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, thermostating 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 thermostating 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 thermostating and barostatting to the system's potential energy as part of [thermodynamic output](#).

The potential energy change due to this fix is stored as a scalar quantity, which can be accessed by various [output commands](#).

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 particles be represented as extended spheres and not point particles. This means they will have an angular velocity and a diameter which is determined by the [shape](#) command.

Any dimension being adjusted by this fix must be periodic. A dimension whose target pressures are specified as NULL can be non-periodic or periodic.

The final Tstop cannot be 0.0 since it would make the target $T = 0.0$ at some timestep during the simulation which is not allowed in the Nose/Hoover formulation.

Related commands:

[fix npt](#), [fix nve_sphere](#), [fix nvt_sphere](#), [fix npt_asphere](#), [fix_modify](#)

Default:

The keyword defaults are `drag = 0.0` and `dilate = all`.

fix nve 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.

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: 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 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 "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 particles be represented as extended ellipsoids and not point particles. This means they will have an angular momentum and a shape which is determined by the [shape](#) command.

Related commands:

[fix nve](#), [fix nve/sphere](#)

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 scalar quantity which 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. This value can be accessed by various [output commands](#). The scalar value calculated by this fix is "extensive", meaning it scales with the number of atoms in the simulation.

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 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: 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 the [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 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 requires that particles be represented as extended spheres and not point particles. This means they will have an angular velocity and a diameter which is determined either by the [shape](#) command or by each particle being assigned an individual radius, e.g. for [atom_style granular](#).

Related commands:

[fix nve](#), [fix nve/asphere](#)

Default: none

fix nvt command

Syntax:

```
fix ID group-ID nvt Tstart Tstop Tdamp keyword value ...
```

- ID, group-ID are documented in [fix](#) command
- nvt = style name of this fix command
- Tstart,Tstop = desired temperature at start/end of run
- Tdamp = temperature damping parameter (time units)
- zero or more keyword/value pairs may be appended
- keyword = *drag*

drag value = drag factor added to thermostat (0.0 = no drag)

Examples:

```
fix 1 all nvt 300.0 300.0 100.0
fix 1 all nvt 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 ([Hoover](#)). V is volume; T is temperature. This creates a system trajectory consistent with the canonical ensemble.

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 from them before thermostating 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).

In some cases (e.g. for solids) the temperature of the system can oscillate undesirably when a Nose/Hoover 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 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.

IMPORTANT NOTE: Unlike the [fix temp/berendsen](#) command which performs thermostating but NO time integration, this fix performs thermostating/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, this fix 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 thermostating.

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 thermostating, 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, thermostating 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 temperature [compute](#) you have defined to this fix which will be used in its thermostating 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 Nose/Hoover thermostating to the system's potential energy as part of [thermodynamic output](#).

The potential energy change due to this fix is stored as a scalar quantity, which can be accessed by various [output commands](#). The scalar value calculated by this fix is "extensive", meaning it scales with the number of atoms in the simulation.

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:

The final Tstop cannot be 0.0 since it would make the target $T = 0.0$ at some timestep during the simulation which is not allowed in the Nose/Hoover formulation.

Related commands:

[fix nve](#), [fix npt](#), [fix temp/rescale](#), [fix langevin](#), [fix_modify](#), [compute temp](#)

Default:

The keyword defaults are $\text{drag} = 0.0$.

(Hoover) Hoover, Phys Rev A, 31, 1695 (1985).

fix nvt/asphere command

Syntax:

```
fix ID group-ID nvt/asphere Tstart Tstop Tdamp keyword value ...
```

- ID, group-ID are documented in [fix](#) command
- nvt/asphere = style name of this fix command
- Tstart,Tstop = desired temperature at start/end of run
- Tdamp = temperature damping parameter (time units)
- zero or more keyword/value pairs may be appended
- keyword = *drag*

drag value = drag factor added to thermostat (0.0 = no drag)

Examples:

```
fix 1 all nvt/asphere 300.0 300.0 100.0
fix 1 all nvt/asphere 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.

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 thermostating 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).

In some cases (e.g. for solids) the temperature of the system can oscillate undesirably when a Nose/Hoover 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 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.

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 thermostating, 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, thermostating 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 thermostating procedure.

The [fix_modify energy](#) option is supported by this fix to add the energy change induced by Nose/Hoover thermostating to the system's potential energy as part of [thermodynamic output](#).

The potential energy change due to this fix is stored as a scalar quantity, which can be accessed by various [output commands](#).

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 particles be represented as extended ellipsoids and not point particles. This means they will have an angular momentum and a shape which is determined by the [shape](#) command.

The final Tstop cannot be 0.0 since it would make the target $T = 0.0$ at some timestep during the simulation which is not allowed in the Nose/Hoover formulation.

Related commands:

[fix nvt](#), [fix nve_asphere](#), [fix npt_asphere](#), [fix_modify](#)

Default:

The keyword defaults are `drag = 0.0`.

fix nvt/sllod command

Syntax:

```
fix ID group-ID nvt/sllod Tstart Tstop Tdamp keyword value ...
```

- ID, group-ID are documented in [fix](#) command
- nvt/sllod = style name of this fix command
- Tstart, Tstop = desired temperature at start/end of run
- Tdamp = temperature damping parameter (time units)
- zero or more keyword/value pairs may be appended
- keyword = *drag*

drag value = drag factor added to thermostat (0.0 = no drag)

Examples:

```
fix 1 all nvt/sllod 300.0 300.0 100.0
fix 1 all nvt/sllod 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 thermostating. 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.

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).

In some cases (e.g. for solids) the temperature of the system can oscillate undesirably when a Nose/Hoover 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 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.

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 thermostating, 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, thermostating 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 thermostating procedure.

The [fix_modify energy](#) option is supported by this fix to add the energy change induced by Nose/Hoover thermostating to the system's potential energy as part of [thermodynamic output](#).

The potential energy change due to this fix is stored as a scalar quantity, which can be accessed by various [output commands](#).

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:

The final Tstop cannot be 0.0 since it would make the target $T = 0.0$ at some timestep during the simulation which is not allowed in the Nose/Hoover formulation.

Related commands:

[fix nve](#), [fix npt](#), [fix npt](#), [fix temp/rescale](#), [fix langevin](#), [fix_modify](#), [compute temp](#)

Default:

The keyword defaults are drag = 0.0.

(Tuckerman) Tuckerman, Mundy, Balasubramanian, Klein, J Chem Phys, 106, 5615 (1997).

fix nvt/sphere command

Syntax:

```
fix ID group-ID nvt/sphere Tstart Tstop Tdamp keyword value ...
```

- ID, group-ID are documented in [fix](#) command
- nvt/sphere = style name of this fix command
- Tstart, Tstop = desired temperature at start/end of run
- Tdamp = temperature damping parameter (time units)
- zero or more keyword/value pairs may be appended
- keyword = *drag*

drag value = drag factor added to thermostat (0.0 = no drag)

Examples:

```
fix 1 all nvt/sphere 300.0 300.0 100.0
fix 1 all nvt/sphere 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 thermostating 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).

In some cases (e.g. for solids) the temperature of the system can oscillate undesirably when a Nose/Hoover 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 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.

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 thermostating, 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, thermostating 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 thermostating procedure.

The [fix_modify energy](#) option is supported by this fix to add the energy change induced by Nose/Hoover thermostating to the system's potential energy as part of [thermodynamic output](#).

The potential energy change due to this fix is stored as a scalar quantity, which can be accessed by various [output commands](#).

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 particles be represented as extended spheres and not point particles. This means they will have an angular velocity and a diameter which is determined by the [shape](#) command.

The final Tstop cannot be 0.0 since it would make the target $T = 0.0$ at some timestep during the simulation which is not allowed in the Nose/Hoover formulation.

Related commands:

[fix nvt](#), [fix nve_sphere](#), [fix nvt_asphere](#), [fix npt_sphere](#), [fix_modify](#)

Default:

The keyword defaults are `drag = 0.0`.

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

$$\xi_i = \sum_{j=1}^{12} |\mathbf{r}_j - \mathbf{r}_j^I| \quad (1)$$

$$\xi_{IJ} = \sum_{j=1}^{12} |\mathbf{r}_j^J - \mathbf{r}_j^I| \quad (2)$$

$$\xi_{\text{low}} = \text{cutlo} \xi_{IJ} \quad (3)$$

$$\xi_{\text{high}} = \text{cuthi} \xi_{IJ} \quad (4)$$

$$\omega_i = \frac{\pi}{2} \frac{\xi_i - \xi_{\text{low}}}{\xi_{\text{high}} - \xi_{\text{low}}} \quad (5)$$

$$\begin{aligned} u_i &= 0 && \text{for } \xi_i < \xi_{\text{low}} \\ &= dE \frac{1 - \cos(2\omega_i)}{2} && \text{for } \xi_{\text{low}} < \xi_i < \xi_{\text{high}} \\ &= dE && \text{for } \xi_{\text{high}} < \xi_i \end{aligned} \quad (6)$$

which are fully explained in (Janssens). The order parameter ξ_i for atom I in equation (1) is a sum over the 12 nearest neighbors of atom I . \mathbf{r}_j is the vector from atom I to its neighbor J , and \mathbf{r}_j^I is a vector in the reference (perfect) crystal. That is, if $\text{dir} = 0/1$, then \mathbf{r}_j^I is a vector to an atom coord from file 0/1. Equation (2) gives the expected value of the order parameter ξ_{IJ} 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 U_i for atom I is given in equation (6) where it is interpolated between 0 and dE using the two threshold ξ_i values and the W_i 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 $1.57 * \text{alat} / \sqrt{2}$ 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](#).

The potential energy change due to this fix is stored as a scalar quantity, which can be accessed by various [output commands](#). The scalar value calculated by this fix is "extensive", meaning it scales with the number of atoms in the simulation.

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	1.7853000000000000	1.596820864092150
0.798410432046075	1.7853000000000000	-1.596820864092150
0.798410432046075	0.0000000000000000	2.395231296138225
2.395231296138225	0.0000000000000000	-0.798410432046075
1.596820864092150	1.7853000000000000	0.798410432046075
1.596820864092150	-1.7853000000000000	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 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.

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 it 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 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 "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 $M = \# \text{ of inserted particles} * N_{\text{attempt}}$. 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 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 "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.

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 p-style args keyword value ...
```

- ID, group-ID are documented in [fix](#) command
- press/berendsen = style name of this fix command
- p-style = *xyz* or *xy* or *yz* or *xz* or *aniso*

```
xyz args = Pstart Pstop Pdamp
          Pstart,Pstop = desired pressure at start/end of run (pressure units)
          Pdamp = pressure damping parameter (time units)
xy or yz or xz args = Px0 Px1 Py0 Py1 Pz0 Pz1 Pdamp
          Px0,Px1,Py0,Py1,Pz0,Pz1 = desired pressure in x,y,z at
          start/end (0/1) of run (pressure units)
          Pdamp = pressure damping parameter (time units)
aniso args = Px0 Px1 Py0 Py1 Pz0 Pz1 Pdamp
          Px0,Px1,Py0,Py1,Pz0,Pz1 = desired pressure in x,y,z at
          start/end (0/1) of run (pressure units)
          Pdamp = pressure damping parameter (time units)
```

- zero or more keyword/value pairs may be appended
- keyword = *dilate* or *modulus*

```
dilate value = all or partial
modulus value = bulk modulus of system (pressure units)
```

Examples:

```
fix 1 all press/berendsen xyz 0.0 0.0 1000.0
fix 2 all press/berendsen aniso 0.0 0.0 0.0 0.0 NULL NULL 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 withing 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 thermostating 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 thermostating and barostatting.

The pressure can be controlled in one of several styles, as specified by the *p-style* argument. In each case, the desired pressure at each timestep is a ramped value during the run from the starting value to the end value.

Style *xyz* means couple all dimensions together when pressure is computed (isotropic pressure), and dilate/contract the dimensions together.

Styles *xy* or *yz* or *xz* means that the 2 specified dimensions are coupled together, both for pressure computation and for dilation/contraction. The 3rd dimension dilates/contracts independently, using its pressure component as the driving force. These styles cannot be used for a 2d simulation.

For style *aniso*, all dimensions dilate/contract independently using their individual pressure components as the driving forces.

For any of the styles except *xyz*, any of the independent pressure components (e.g. *z* in *xy*, or any dimension in *aniso*) can have their target pressures (both start and stop values) specified as NULL. This means that no pressure control is applied to that dimension so that the box dimension remains unchanged. For a 2d simulation the *z* pressure components must be specified as NULL when using style *aniso*.

For styles *xy* and *yz* and *xz*, the starting and stopping pressures must be the same for the two coupled dimensions and cannot be specified as NULL.

In some cases (e.g. for solids) the pressure (volume) and/or temperature of the system can oscillate undesirably when a Nose/Hoover barostat 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 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.

For all pressure styles, the simulation box stays rectangular in shape. Parinello–Rahman boundary condition for tilted boxes (triclinic symmetry) are supported by other LAMMPS commands (see [this section](#) of the manual), but not yet by this command.

For all styles, 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.

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 scalar or vector 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. A dimension whose target pressures are specified as NULL can be non-periodic or 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 *screen*

```
file value = filename  
screen value = yes or no
```

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* keyword is used, a filename is specified to which the output generated by this fix will be written.

If the *screen* keyword is used, output by this fix to the screen and logfile can be turned on or off as desired.

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: none

Related commands:

[variable](#), [print](#)

Default:

The option defaults are no file output and screen = yes.

fix rdf command

Syntax:

```
fix ID group-ID rdf N file Nbin itype1 jtype1 itype2 jtype2 ...
```

- ID, group-ID are documented in [fix](#) command
- rdf = style name of this fix command
- N = compute radial distribution function (RDF) every this many timesteps
- file = filename to write radial distribution function info to
- Nbin = number of RDF bins
- itypeN = central atom type for RDF pair N
- jtypeN = distribution atom type for RDF pair N

Examples:

```
fix 1 all rdf 500 rdf.out 100 1 1
fix 1 fluid rdf 10000 rdf.out 100 1 1 1 2 2 1 2 2
```

Description:

Compute the radial distribution function (RDF), also known as $g(r)$, and coordination number every N steps. The RDF for each specified atom type pair is histogrammed in Nbin bins from distance 0 to R_c , where R_c = the maximum force cutoff for any pair of atom types. An atom pair only contributes to the RDF if

- both atoms are in the fix group
- the distance between them is within the maximum force cutoff
- their interaction is stored in the neighbor list

Bonded atoms (1–2, 1–3, 1–4 interactions within a molecular topology) with a pairwise weighting factor of 0.0 are not included in the RDF; pairs with a non-zero weighting factor are included. The weighting factor is set by the [special_bonds](#) command.

The RDF statistics for each timestep are written to the specified file, as are the RDF values averaged over all timesteps.

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:

The RDF is not computed for distances longer than the force cutoff, since processors (in parallel) don't know atom coordinates for atoms further away than that distance. If you want an RDF for larger r , you'll need to post-process a dump file.

Related commands:

pair_style

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](#) thermostating).

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 thermostating 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 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 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 rigid

Syntax:

```
fix ID group-ID rigid keyword values
```

- ID, group-ID are documented in [fix](#) command
- rigid = style name of this fix command
- keyword = *single* or *molecule* or *group*

```
single values = none  
molecule values = none  
group values = list of group IDs
```

Examples:

```
fix 1 clump rigid single  
fix 1 polychains rigid molecule  
fix 2 fluid rigid group clump1 clump2 clump3
```

Description:

Treat one or more sets of atoms as an independent rigid body. This means that each timestep the total force and torque on each rigid body is computed and the coordinates and velocities of the atoms in each body are updated so that they move as a rigid body. This can be useful for freezing one or more portions of a large biomolecule, or for simulating a system of colloidal particles.

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).

Each body must have two or more atoms. Which atoms are in which bodies can be defined via several options.

For option *single* the entire group of atoms is treated as one 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 *group*, each of the listed groups is treated as a separate rigid body. Note that only atoms that are also in the fix group are included in each rigid body.

For computational efficiency, you should also 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 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.

This fix uses constant-energy integration, so you may need to impose additional constraints to control the temperature of an ensemble of rigid bodies. You can use [fix langevin](#) for this purpose to treat the system as effectively immersed in an implicit solvent, i.e. a Brownian dynamics model. Or you can thermostat additional atoms of an explicit solvent directly.

The degrees-of-freedom removed by rigid bodies are accounted for in temperature and pressure computations. Similarly, the rigid body contribution to the pressure of the system (virial) is also accounted for. For linear rigid bodies of three or more atoms, one additional degree-of-freedom must be subtracted manually using the `compute_modify` command. E.g. for a simulation of 10 such rigid bodies, use "compute_modify thermo_temp extra 13", after the thermo_style command, where 3 is the default setting and an additional 10 degrees-of-freedom are subtracted.

IMPORTANT NOTE: The periodic image flags of atoms in rigid bodies are modified when the center-of-mass of the rigid body moves across a periodic boundary. They are not incremented/decremented as they would be for non-rigid atoms. This change does not affect dynamics, but means that any diagnostic computation based on the atomic image flag values must be adjusted accordingly. For example, the `fix msd` will not compute the expected mean-squared displacement for such atoms, and 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. 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.

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 performs an MPI_Allreduce each timestep that is proportional in length to the number of rigid bodies. Hence it will not scale well in parallel if large numbers of rigid bodies are simulated.

If the atoms in a single rigid body initially straddle a periodic boundary, the input data file must define the image flags for each atom correctly, so that LAMMPS can "unwrap" the atoms into a valid rigid body.

You should not use this fix if you just want to hold group of atoms stationary. A better way to do this 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.

Related commands:

`delete_bonds`, `neigh_modify` exclude

Default: none

fix setforce command

Syntax:

```
fix ID group-ID setforce fx fy fz
```

- ID, group-ID are documented in [fix](#) command
- setforce = style name of this fix command
- fx,fy,fz = force component values

Examples:

```
fix freeze indenter setforce 0.0 0.0 0.0  
fix 2 edge setforce NULL 0.0 0.0
```

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, assuming their initial velocity zero.

Any of the fx,fy,fz values can be specified as NULL which means do not alter the force component in that dimension.

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 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", meaning they scale with the number of atoms in the simulation.

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 addforce](#), [fix aveforce](#)

Default: none

fix shake 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. $1.0\text{e-}4 = 1$ 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.

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:

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 $K (R - R0) M_i / M$ is applied to each atom in the group where *K* is the spring constant, *M_i* 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", meaning they scale with the number of atoms in the simulation.

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-smd" 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 $K (R - R0) M_i / M$ is applied to each atom in the group where K is the spring constant, M_i 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](#). None of the [fix_modify](#) options are relevant to this fix.

This fix computes a 3-vector of forces, which can be accessed by various [output commands](#). This 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).

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 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.

$$R_G^2 = \frac{1}{M} \sum_i^N m_i \left(x_i - \frac{1}{M} \sum_j^N m_j x_j \right)^2$$

$$E = K (R_G - R_{G0})^2$$

$$F_i = 2K \frac{m_i}{M} \left(1 - \frac{R_{G0}}{R_G} \right) \left(x_i - \frac{1}{M} \sum_j^N m_j x_j \right)$$

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 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: 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 scalar energy which can be accessed by various [output commands](#). This energy is the sum of the spring energy for each atom, where the per-atom energy is $0.5 * K * r^2$.

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 temp/berendsen 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 nvt 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 thermostating 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 thermostating AND time integration, this fix does NOT perform time integration. It only modifies velocities to effect thermostating. 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 thermostating.

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 thermostating, 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, thermostating 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:

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 thermostating procedure, as described above. For consistency, the group used by this fix and by the compute should be the same.

No global scalar or vector or per-atom quantities are stored by this fix for access by various [output commands](#).

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

Syntax:

```
fix ID group-ID temp/rescale N Tstart Tstop window fraction keyword values ...
```

- 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 thermostating 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 thermostating AND time integration, this fix does NOT perform time integration. It only modifies velocities to effect thermostating. 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 thermostating.

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 thermostating, 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, thermostating 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:

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 thermostating 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](#). Note that because this fix is invoked every N steps and thermodynamic info is printed every M steps, that unless M is a multiple of N, the energy contribution will be zero.

The potential energy change due to this fix is stored as a scalar quantity, which can be accessed by various [output commands](#). The scalar value calculated by this fix is "extensive", meaning it scales with the number of atoms in the simulation.

The energy change can be printed as part of thermodynamic output via the keyword *f_ID*, where ID is the fix-ID of this fix. See the [thermo_style custom](#) command for details.

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 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.

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 appropriate 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.

The cumulative 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 accumulates thereafter, once every *N* steps. The units of the quantity are energy; see the [units](#) command for details. This quantity can be accessed by various [output commands](#), such as [thermo_style custom](#). The scalar value calculated by this fix is "intensive", meaning it is independent of the number of atoms in the simulation.

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:

LAMMPS does not check, but the masses of all exchanged atom pairs should be the same to use this fix in a way that conserves both momentum and kinetic energy. 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 between solvent particles.

Related commands:

[fix ave/spatial](#), [fix viscosity](#)

Default:

The option defaults are *swap* = 1.

(**Muller–Plathe**) Muller–Plathe and Reith, Computational and Theoretical Polymer Science, 9, 203–209 (1999).

(**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 scalar or vector 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 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 $N_{bin}/2 + 1$ layer. Thus if $N_{bin} = 20$, 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 N_{bin} 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 appropriate 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 simulation 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.

The cumulative 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 accumulates thereafter, once every `N` steps. The units of the quantity are momentum = mass*velocity. This quantity can be accessed by various [output commands](#), such as [thermo_style custom](#). The scalar value calculated by this fix is "intensive", meaning it is independent of the number of atoms in the simulation.

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:

If the masses of all exchange partners are the same, then swaps conserve both momentum and kinetic energy. 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 between solvent particles.

Related commands:

[fix ave/spatial](#), [fix thermal/conductivity](#)

Default:

The option defaults are $\text{swap} = 1$ and $\text{vtarget} = \text{INF}$.

(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

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 can be appended
- keyword = *b* or *a* or *t* or *m*
- 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. In granular simulations this can be useful for draining the kinetic energy from the system in a controlled fashion. If used without additional thermostating (to add kinetic energy to the system), it has the effect of slowly (or rapidly) freezing the system; hence it is a simple energy minimization technique.

The damping force F is given by $F = -\text{gamma} * \text{velocity}$. 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.

In a Brownian dynamics context, $\text{gamma} = kT / mD$, where k = Boltzmann's constant, T = temperature, m = particle mass, and D = particle diffusion coefficient. D can be written as $kT / (6 \pi \eta d)$, where η = viscosity of the frictional fluid and d = diameter of particle. This means $\text{gamma} = 6 \pi \eta d$, 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 (in centiPoise or some other units), 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 adds both a viscous damping term and this random force to each atom; hence if using *fix langevin* you do not typically need to use *fix viscous*.

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: none

Related commands:

[fix langevin](#)

Default: none

fix wall/gran command

Syntax:

```
fix ID group-ID wall/gran wallstyle args keyword values ...
```

- ID, group-ID are documented in [fix](#) command
- wall/gran = style name of this fix command
- style = *xplane* or *yplane* or *zplane* or *zcylinder*
- args = list of arguments for a particular style

```
xplane or yplane or zplane args = lo hi gamma xmu
    lo,hi = position of lower and upper plane (distance units), either can be NULL)
    gamman = damping coeff for normal direction collisions with wall
    xmu = friction coeff for the wall
zcylinder args = radius gamma xmu
    radius = cylinder radius (distance units)
    gamman = damping coeff for normal direction collisions with wall
    xmu = friction coeff for the wall
```

- 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 xplane -10.0 10.0 50.0 0.5
fix 2 all wall/gran zcylinder 15.0 50.0 0.5 wiggle z 3.0 2.0
fix 1 all wall/gran zplane 0.0 NULL 100.0 0.5
```

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 *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. For all wallstyles, a damping and friction coefficient for particle–wall interactions are also 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 frozen particles specified by the [fix_wiggle](#) 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 its *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 = pos0 + A - A cos (omega * delta)
```

where *pos0* is the position at the time the fix was specified, *A* is the *amplitude*, *omega* is $2\text{ PI} / \text{period}$, and *delta* is the elapsed time 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 its 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 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 "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_wiggle](#)

Default: none

fix wall/lj126 command

Syntax:

```
fix ID group-ID wall/lj126 style coord epsilon sigma cutoff keyword values ...
```

- ID, group-ID are documented in [fix](#) command
- wall/lj126 = style name of this fix command
- style = *xlo* or *xhi* or *ylo* or *yhi* or *zlo* or *zhi*
- coord = position of wall
- epsilon = Lennard–Jones epsilon for wall–particle interaction
- sigma = Lennard–Jones sigma for wall–particle interaction
- cutoff = distance from wall at which wall–particle interaction is cut off (distance units)
- one or more keyword/value pairs may be appended
- keyword = *vel*

```
vel args = v
v = velocity of wall in perpendicular direction (velocity units)
```

Examples:

```
fix wallhi all wall/lj126 xhi 10.0 1.0 1.0 1.12
fix leftwall all wall/lj126 zlo 0.0 1.0 1.0 0.858 vel 1.0
```

Description:

Bound the simulation domain on one of its faces with a Lennard–Jones wall that interacts with the atoms in the group. The energy E of wall–particle interactions is given by the 12–6 potential

$$E = 4\epsilon \left[\left(\frac{\sigma}{r} \right)^{12} - \left(\frac{\sigma}{r} \right)^6 \right] \quad r < r_c$$

where r is the distance from the particle to the wall *coord*, and epsilon and sigma are the usual LJ parameters. r_c is the cutoff value specified in the command. This interaction provides a harder, more repulsive interaction with the wall than the softer 9–3 potential provided by the [fix wall/lj93](#) command.

The wall potential is shifted so that the energy of a wall–particle interaction is 0.0 at the cutoff distance.

If the *vel* keyword is specified, the position of wall will move during the simulation, based on its initial position (coord) and the specified velocity (vel). *Xlo* or *xhi* walls move in the x direction with whatever sign you give the *v* argument. Ditto for y and z walls. Note that if you do multiple runs, the initial position of the wall does not change, so it will continue to move at the specified velocity.

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 scalar energy and a 3–vector of forces (on the wall), which can be accessed by various [output commands](#). The scalar and vector values calculated by this fix are "extensive", meaning they scale with the number of atoms in the simulation.

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 LJ 12/6 wall must be non–periodic.

Related commands:

[fix wall/reflect](#), [fix wall/lj93](#)

Default:

The option defaults are $vel = 0$.

fix wall/lj93 command

Syntax:

```
fix ID group-ID wall/lj93 style coord epsilon sigma cutoff keyword values ...
```

- ID, group-ID are documented in [fix](#) command
- wall/lj93 = style name of this fix command
- style = *xlo* or *xhi* or *ylo* or *yhi* or *zlo* or *zhi*
- coord = position of wall
- epsilon = Lennard–Jones epsilon for wall–particle interaction
- sigma = Lennard–Jones sigma for wall–particle interaction
- cutoff = distance from wall at which wall–particle interaction is cut off (distance units)
- one or more keyword/value pairs may be appended
- keyword = *vel*

```
vel args = v
```

```
v = velocity of wall in perpendicular direction (velocity units)
```

Examples:

```
fix wallhi all wall/lj93 xhi 10.0 1.0 1.0 2.5
fix leftwall all wall/lj93 zlo 0.0 1.0 1.0 0.858 vel 1.0
```

Description:

Bound the simulation domain on one of its faces with a Lennard–Jones wall that interacts with the atoms in the group. The energy E of wall–particle interactions is given by the 9–3 potential

$$E = \epsilon \left[\frac{2}{15} \left(\frac{\sigma}{r} \right)^9 - \left(\frac{\sigma}{r} \right)^3 \right] \quad r < r_c$$

where r is the distance from the particle to the wall *coord*, and epsilon and sigma are the usual LJ parameters. R_c is the cutoff value specified in the command. This interaction is derived by integrating over a 3d half–lattice of Lennard–Jones 12–6 particles. A harder, more repulsive wall interaction can be computed by using the [fix wall/lj126](#) command.

The wall potential is shifted so that the energy of a wall–particle interaction is 0.0 at the cutoff distance.

If the *vel* keyword is specified, the position of wall will move during the simulation, based on its initial position (coord) and the specified velocity (vel). *Xlo* or *xhi* walls move in the x direction with whatever sign you give the v argument. Ditto for y and z walls. Note that if you do multiple runs, the initial position of the wall does not change, so it will continue to move at the specified velocity.

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 scalar energy and a 3–vector of forces (on the wall), which can be accessed by various `output commands`. The scalar and vector values calculated by this fix are "extensive", meaning they scale with the number of atoms in the simulation.

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 LJ 9/3 wall must be non–periodic.

Related commands:

`fix wall/reflect`, `fix wall/lj126`

Default:

The option defaults are `vel = 0`.

fix wall/reflect command

Syntax:

```
fix ID group-ID wall/reflect keyword ...
```

- ID, group-ID are documented in [fix](#) command
- wall/reflect = style name of this fix command
- one or more keyword/value pairs may be appended
- keyword = *xlo* or *xhi* or *ylo* or *yhi* or *zlo* or *zhi*

Examples:

```
fix xwalls all wall/reflect xlo xhi
fix walls all wall/reflect xlo ylo zlo xhi yhi zhi
```

Description:

Bound the simulation with one or more walls which reflect particles when they attempt to move thru them.

Reflection means that if an atom moves outside the box on a timestep by a distance delta (e.g. due to [fix nve](#)), then it is put back inside the box 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.

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:

Any dimension (xyz) that has a reflecting wall must be non–periodic.

A reflecting wall cannot 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](#).

Related commands:

[fix wall/lj93](#) command

Default: none

(Bond) Bond and Leimkuhler, SIAM J Sci Comput, 30, p 134 (2007).

fix wiggle command

Syntax:

fix ID group-ID wiggle dim amplitude period

- ID, group-ID are documented in [fix](#) command
- wiggle = style name of this fix command
- dim = *x* or *y* or *z*
- amplitude = size of oscillation (distance units)
- period = time of oscillation (time units)

Examples:

```
fix 1 frozen wiggle y 3.0 0.5
```

Description:

Move a group of atoms in a sinusoidal oscillation. This is useful in granular simulations when boundary atoms are wiggled to induce packing of the dynamic atoms. The dimension *dim* of movement is specified as is the *amplitude* and *period* of the oscillations. Each timestep the *dim* coordinate of each atom is set to

$$\text{coord} = \text{coord0} + A - A \cos(\omega * \text{delta})$$

where *coord0* is the coordinate at the time the fix was specified, *A* is the *amplitude*, *omega* is $2 \text{ PI} / \text{period}$, and *delta* is the elapsed time since the fix was specified. The velocity of the atom is set to the derivative of this expression.

Note that if a value of $A > 0$ is specified, each wiggling atom does not oscillate around its initial position, but moves entirely in the positive direction relative to its initial position. This is so that its initial velocity will be 0.0.

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: none

Related commands: none

Default: none

group command

Syntax:

```
group ID style args
```

- ID = user-defined name of the group
- style = *region* or *type* or *id* or *molecule* or *subtract* or *union* or *intersect*

```

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

```

Description:

Identify a collection of atoms as belonging to a group. The group ID can then be used in other commands such as `fix`, `velocity`, `dump`, or `temperature` to act on the atoms together.

If the group ID already exists, the group command adds the specified atoms to the group.

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.

Restrictions:

There can be no more than 32 defined groups, including "all".

Related commands:

region, fix, velocity, dump, temperature

Default:

All atoms belong to the "all" group.

if command

Syntax:

```
if value1 operator value2 then command1 else command2
```

- value1 = 1st value
- operator = "" or ">=" or "==" or "!="
- value2 = 2nd value
- then = required word
- command1 = command to execute if condition is met
- else = optional word
- command2 = command to execute if condition is not met (optional argument)

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 then "timestep 0.005"
if ${eng} > ${eng_previous} then "jump file1" else "jump file2"
```

Description:

This command provides an in–then–else test capability within an input script. Two values are numerically compared to each other and the result is TRUE or FALSE. Note that as in the examples above, either of the values can be variables, as defined by the [variable](#) command, so that when they are evaluated when substituted for in the if command, a user–defined computation will be performed which can depend on the current state of the simulation.

If the result of the if test is TRUE, then command1 is executed. This can be any valid LAMMPS input script command. If the command is more than 1 word, it should be enclosed in double quotes, so that it will be treated as a single argument, as in the examples above.

The if command can contain an optional "else" clause. If it does and the result of the if test is FALSE, then command2 is executed.

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](#) for info on how to delete the associated loop variable, so that it can be re–used later in the input script.

Note that if either command1 or command2 is a bogus LAMMPS command, such as "exit" in the first example, then executing the command will cause LAMMPS to halt.

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
variable  b delete

next      a
jump      in.script loopa
```

Restrictions: none

Related commands:

[variable](#)

Default: none

improper_style class2 command

Syntax:

```
improper_style class2
```

Examples:

```
improper_style class2
improper_coeff 1 100.0 0
```

Description:

The *class2* improper style uses the potential

$$\begin{aligned}
 E &= E_i + E_{aa} \\
 E_i &= K \left[\frac{\chi_{ijkl} + \chi_{kjli} + \chi_{ljik}}{3} - \chi_0 \right]^2 \\
 E_{aa} &= M_1(\theta_{ijk} - \theta_1)(\theta_{kjl} - \theta_3) + \\
 &\quad M_2(\theta_{ijk} - \theta_1)(\theta_{ijl} - \theta_2) + \\
 &\quad M_3(\theta_{ijl} - \theta_2)(\theta_{kjl} - \theta_3)
 \end{aligned}$$

where E_i is the improper term and E_{aa} is an angle–angle term. The 3 χ terms in E_i 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. χ_{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 χ_{KJLI} and χ_{LJIK} . Note that atom J appears in the common bonds (JI, JK, JL) of all 3 χ terms. Thus J (the 2nd atom in the quadruplet) is the atom of symmetry in the 3 χ 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. θ_{IJL} is the angle formed by atoms I,J,L with J in the middle. θ_1 , θ_2 , θ_3 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.

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:

For this style, coefficients for the E_i formula can be specified in either the input script or data file. These are the 2 coefficients:

- K (energy/radian²)
- χ_0 (degrees)

X0 is specified in degrees, but LAMMPS converts it to radians internally; hence the units of K are in energy/radian².

Coefficients for the Eaa formula can only be specified in the data file. For the Eaa formula, the coefficients are listed under a "AngleAngle Coeffs" heading and each line lists 6 coefficients:

- 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/radian².

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
```

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

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 ϕ 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 χ is the improper angle, χ_0 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 χ 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 χ 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/radian²)
- χ_0 (degrees)

χ_0 is specified in degrees, but LAMMPS converts it to radians internally; hence the units of K are in energy/radian².

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 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

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
```

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
```

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.

Optionally, if a 2nd argument is used, 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 ppm. 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 ppm. 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 interpolating 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 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 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.

Restrictions: none

Related commands:

[kspace_style](#), [boundary](#)

Default:

The option defaults are mesh = 0 0 0, order = 5, geweld = 0.0, and slab = 1.0.

kspace_style command

Syntax:

```
kspace_style style value
```

- style = *none* or *ewald* or *pppm* or *pppm/tip4p* or *ewald/n*

```
none value = none
ewald value = precision
precision = desired accuracy
pppm value = precision
precision = desired accuracy
pppm/tip4p value = precision
precision = desired accuracy
ewald/n value = precision
precision = desired accuracy
```

Examples:

```
kspace_style ppm 1.0e-4
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 $N \log(N)$ due to the FFTs, so it is almost always a faster choice ([Pollock](#)).

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.

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 $1.0\text{e-}4$ 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.

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

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 *hcp* 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 a_1, a_2, a_3 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 $a_1 = 1\ 0\ 0$, $a_2 = 0\ 1\ 0$, and $a_3 = 0\ 0\ 1$. Style *hcp* has $a_1 = 1\ 0\ 0$, $a_2 = 0\ \sqrt{3}\ 0$, and $a_3 = 0\ 0\ \sqrt{8/3}$. 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 $a_1 = 1\ 0\ 0$ and $a_2 = 0\ 1\ 0$. 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 $a_1 = 1\ 0\ 0$ and $a_2 = 0\ \sqrt{3}\ 0$. 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 a_1 , a_2 , a_3 , and a list of basis atoms to put in the unit cell. By default, a_1 and a_2 and a_3 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 ($0.0 \leq x, y, z < 1.0$), 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 ρ^* . LAMMPS converts this value into the multiplicative factor via the formula " $\text{factor}^{\text{dim}} = \rho / \rho^*$ ", 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 $\text{dim} = 2$ 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 ($0.0 \leq x, y, z < 1.0$) 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. ($X \text{ cross } Y$) 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 a_1, a_2, a_3 vectors, then think of the 3 *orient* options as creating a 3x3 rotation matrix which is applied to a_1, a_2, a_3 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 a_1, a_2, a_3 . Thus a *hex* style lattice with a scale factor of 3.0 Angstroms, would have a lattice spacing of 3.0 in x and $3 \cdot \sqrt{3}$ 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, $a_1 = 1$ 0 0, $a_2 = 0$ 1 0, and $a_3 = 0$ 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. Mass values can also be set in the [read_data](#) data file. See the [units](#) command for what mass units to use.

Most atom styles require per-type masses to be specified. One exception is [atom_style granular](#) or [atom_style peri](#), where masses are defined for individual atoms, not types. These are defined in the file read by the [read_data](#) command, or set to default values by the [create_atoms](#) command, or set to new values by the [set density](#) command. Also note that [pair_style eam](#) defines the masses of atom types in the EAM potential file.

I 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 specifies mass uses 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
```

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 (if the atom style requires masses be set). They must also all be defined before a [velocity](#) or [fix shake](#) command is used.

Related commands: none

Default: none

min_modify command

Syntax:

```
min_modify keyword values ...
```

- one or more keyword/value pairs may be listed keyword = *dmax* *dmax* value = max max = maximum distance for line search to move (distance units)

Examples:

```
min_modify dmax 0.2
```

Description:

This command sets parameters that affect the energy minimization algorithms. The various settings may effect 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 minimization algorithms have an outer iteration (conjugate gradient or steepest descent) 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). Thus a value of 0.1 in real distance units means no atom will move further than 0.1 Angstroms in a single outer iteration. This is typically set to avoid the possibility than one atom will be moved through another due to strong overlapping forces.

Restrictions: none

Related commands:

[min_style](#), [minimize](#)

Default:

The option defaults are *dmax* = 0.1.

min_style command

Syntax:

```
min_style style
```

- style = *cg* or *sd*

Examples:

```
min_style cg  
min_style sd
```

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.

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.

Restrictions: none

Related commands:

[min_modify](#), [minimize](#)

Default:

```
min_style cg
```

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 total 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.

A minimization 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 evaluations 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 objective function being minimized is the total potential energy of the system as a function of the N atom coordinates:

$$E(r_1, r_2, \dots, r_N) = \sum_{i,j} E_{pair}(r_i, r_j) + \sum_{ij} E_{bond}(r_i, r_j) + \sum_{ijk} E_{angle}(r_i, r_j, r_k) + \\ \sum_{ijkl} E_{dihedral}(r_i, r_j, r_k, r_l) + \sum_{ijkl} E_{improper}(r_i, r_j, r_k, r_l) + \sum_i E_{fix}(r_i)$$

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 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 10⁴.

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.

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 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.

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: It is highly recommended that you 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 this is not done, 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.

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 individual [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 *Astap* value is used for the prefactor (no time dependence).

Only some 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.

The volume of the simulation domain is not allowed to change during a minimization. Ideally we would allow a fix such as *npt* to impose an external pressure that would be included in the minimization (i.e. allow the box dimensions to change), but this has not yet been implemented.

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

neigh_modify command

Syntax:

neigh_modify keyword values ...

- one or more keyword/value pairs may be listed

```
keyword = delay or every or check 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
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 residuel chain3
neigh_modify exclude molecule rigid
```

Description:

This command sets parameters that affect the building and use of pairwise neighbor lists.

The *every*, *delay*, and *check* 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.

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.

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 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, 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. *Atom*- or *equal*- or *world*-style variables cannot be incremented by a next command. All the variables specified are incremented by one value from their respective lists.

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
```

pair_style airebo command

Syntax:

```
pair_style airebo cutoff LJ_flag TORSION_flag
```

- cutoff = LJ cutoff (sigma scale factor)
- LJ_flag = 0/1 to turn off/on the LJ term in AIREBO (optional)
- TORSION_flag = 0/1 to turn off/on the torsion term in AIREBO (optional)

Examples:

```
pair_style airebo 3.0
pair_style airebo 2.5 1 0
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. The potential consists of three terms:

$$E = \frac{1}{2} \sum_i \sum_{j \neq i} \left[E_{ij}^{REBO} + E_{ij}^{LJ} + \sum_{k \neq i, j} \sum_{l \neq i, j, k} E_{ijkl}^{TORSION} \right]$$

By default, all three terms are included. 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.

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 LJ 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 ($2 < r < \text{cutoff}$) 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* style which specifies an AIREBO potential file with parameters for C and H. 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. Also note that the E_LJ and E_TORSION terms in AIREBO are intended to be used with the E_REBO term and not as stand-alone potentials. Thus we don't suggest you use `pair_style airebo` with the E_REBO term turned off.

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 "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 potential requires 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 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 buck command

pair_style buck/coul/cut command

pair_style buck/coul/long 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

$$E = Ae^{-r/\rho} - \frac{C}{r^6} \quad r < r_c$$

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.

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–distance⁶ 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 LJ and Coulombic cutoffs 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 *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.

Mixing, shift, table, tail correction, restart, rRESPA info:

None of the Buckingham pair styles support mixing. Thus, coefficients for all I,J pairs must be specified explicitly.

All of the Buckingham pair styles support the `pair_modify` shift option for the energy of the `exp()` and `1/r6` 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.

None of the Buckingham pair styles support the `pair_modify` tail option for adding long–range tail corrections to energy and pressure.

All of the Buckingham 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 Buckingham 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 *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](#)

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

$$E = Ae^{-r/\rho} - \frac{C}{r^6} \quad r < r_c$$

$$E = \frac{Cq_iq_j}{\epsilon r} \quad r < r_c$$

R_c 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_lj` 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_lj` 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_lj` 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-distance⁶ 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 style 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/r⁶ 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 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](#)

(Ismail) Ismail, Tsige, In 't Veld, Grest, Molecular Physics (accepted) (2007).

pair_style lj/charmm/coul/charmm command

pair_style lj/charmm/coul/charmm/implicit command

pair_style lj/charmm/coul/long 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` or `lj/charmm/coul/long/opt`
- `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/opt 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.

$$\begin{aligned}
E &= LJ(r) & r < r_{\text{in}} \\
&= S(r) * LJ(r) & r_{\text{in}} < r < r_{\text{out}} \\
&= 0 & r > r_{\text{out}} \\
E &= C(r) & r < r_{\text{in}} \\
&= S(r) * C(r) & r_{\text{in}} < r < r_{\text{out}} \\
&= 0 & r > r_{\text{out}} \\
LJ(r) &= 4\epsilon \left[\left(\frac{\sigma}{r} \right)^{12} - \left(\frac{\sigma}{r} \right)^6 \right] \\
C(r) &= \frac{C q_i q_j}{\epsilon r} \\
S(r) &= \frac{[r_{\text{out}}^2 - r^2]^2 [r_{\text{out}}^2 + 2r^2 - 3r_{\text{in}}^2]}{[r_{\text{out}}^2 - r_{\text{in}}^2]^3}
\end{aligned}$$

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 is 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.

Style `lj/charmm/coul/long/opt` is an optimized version of style `lj/charmm/coul/long` that should give identical answers. Depending on system size and the processor you are running on, it may be 5–25% faster (for the pairwise portion of the run time).

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:

- 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.

Mixing, shift, table, tail correction, restart, rRESPA info:

For atom type pairs I, J and $I \neq J$, the `epsilon`, `sigma`, `epsilon_14`, and `sigma_14` coefficients for all of the `lj/charmm` pair styles can be mixed. They are always mixed with the value *arithmetic*. 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` and `lj/charmm/coul/long/opt` pair styles support the `pair_modify` table option since they 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 "k-space" package. The `lj/charmm/coul/long/opt` style is part of the "opt" package and also requires the "k-space" package. They are only enabled if LAMMPS was built with those package(s) (molecule and kspace are by default). See the [Making LAMMPS](#) section for more info.

On some 64–bit machines, compiling with `-O3` appears to break the Coulombic tabling option used by the `lj/charmm/coul/long` style. See the "Additional build tips" section of the Making LAMMPS documentation pages for workarounds on this issue.

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/coul/cut command

pair_style lj/class2/coul/long 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

$$E = \epsilon \left[2 \left(\frac{\sigma}{r} \right)^9 - 3 \left(\frac{\sigma}{r} \right)^6 \right] \quad r < r_c$$

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.

: line

If the *pair_coeff* command is not used to define coefficients for a particular $I \neq J$ 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.

Mixing, shift, table, tail correction, restart, rRESPA info:

For atom type pairs I,J and $I \neq J$, 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:

All of these pair 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/coul/cut command

pair_style cg/cmm/coul/long 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

$$\begin{aligned}
 E &= \frac{27}{4}\epsilon \left[\left(\frac{\sigma}{r} \right)^9 - \left(\frac{\sigma}{r} \right)^6 \right] & r < r_c \\
 E &= \frac{3\sqrt{3}}{2}\epsilon \left[\left(\frac{\sigma}{r} \right)^{12} - \left(\frac{\sigma}{r} \right)^4 \right] & r < r_c \\
 E &= 4\epsilon \left[\left(\frac{\sigma}{r} \right)^{12} - \left(\frac{\sigma}{r} \right)^6 \right] & r < r_c
 \end{aligned}$$

as required for the CMM Coarse–grained MD parametrization discussed in [\(Shinoda\)](#). R_c is the cutoff.

Style *cg/cmm/coul/cut* adds a Coulombic pairwise interaction given by

$$E = \frac{Cq_iq_j}{\epsilon r} \quad r < r_c$$

where C is an energy–conversion constant, Q_i and Q_j 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

$$E = \frac{Cq_iq_j}{\epsilon r} \exp(-\kappa r) \quad r < r_c$$

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.

Mixing, shift, table, tail correction, restart, and rRESPA info:

For atom type pairs I,J and $I \neq J$, 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. They are only enabled if LAMMPS was built with that package. The `cg/cmm/coul/long` style also requires the "kspace" package to be built (which is enabled by default). See the [Making LAMMPS](#) section for more info.

On some 64–bit machines, compiling with `–O3` appears to break the Coulombic tabling option used by the `cg/cmm/coul/long` style. See the "Additional build tips" section of the Making LAMMPS documentation pages for workarounds on this issue.

Related commands:

[pair_coeff](#), [angle_style cg/cmm](#)

Default: none

(Shinoda) Shinoda, DeVane, Klein, Mol Sim, 33, 27 (2007).

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. $I \leq J$ 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 $I \leq J$ 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 $I \neq J$ 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 airebo](#) – AI-REBO potential
- [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 eam](#) – embedded atom method (EAM)
- [pair_style eam/opt](#) – optimized version of EAM
- [pair_style eam/alloy](#) – alloy EAM
- [pair_style eam/alloy/opt](#) – optimized version of alloy EAM
- [pair_style eam/fs](#) – Finnis–Sinclair EAM
- [pair_style eam/fs/opt](#) – optimized version of Finnis–Sinclair EAM
- [pair_style gayberne](#) – Gay–Berne ellipsoidal potential
- [pair_style gran/hertzian](#) – granular potential with Hertzian interactions
- [pair_style gran/history](#) – granular potential with history effects
- [pair_style gran/no_history](#) – granular potential without history effects
- [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/charmm/coul/long/opt](#) – optimized version of 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/opt](#) – optimized version of cutoff LJ
- [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 lubricate](#) – hydrodynamic lubrication forces
- [pair_style meam](#) – modified embedded atom method (MEAM)
- [pair_style morse](#) – Morse potential
- [pair_style morse/opt](#) – optimized version of Morse potential
- [pair_style peri/pmb](#) – peridynamic PMB potential
- [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

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](#).

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

$$\begin{aligned}
 U_A &= -\frac{A_{cc}}{6} \left[\frac{2a_1a_2}{r^2 - (a_1 + a_2)^2} + \frac{2a_1a_2}{r^2 - (a_1 - a_2)^2} + \ln \left(\frac{r^2 - (a_1 + a_2)^2}{r^2 - (a_1 - a_2)^2} \right) \right] \\
 U_R &= \frac{A_{cc}}{37800} \frac{\sigma^6}{r} \left[\frac{r^2 - 7r(a_1 + a_2) + 6(a_1^2 + 7a_1a_2 + a_2^2)}{(r - a_1 - a_2)^7} \right. \\
 &\quad + \frac{r^2 + 7r(a_1 + a_2) + 6(a_1^2 + 7a_1a_2 + a_2^2)}{(r + a_1 + a_2)^7} \\
 &\quad - \frac{r^2 + 7r(a_1 - a_2) + 6(a_1^2 - 7a_1a_2 + a_2^2)}{(r + a_1 - a_2)^7} \\
 &\quad \left. - \frac{r^2 - 7r(a_1 - a_2) + 6(a_1^2 - 7a_1a_2 + a_2^2)}{(r - a_1 + a_2)^7} \right] \\
 U &= U_A + U_R, \quad r < r_c
 \end{aligned}$$

A_{cc} is the Hamaker constant, a₁ and a₂ are the radii of the two colloidal particles, and R_c 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

$$U = \frac{2 a^3 \sigma^3 A_{cs}}{9 (a^2 - r^2)^3} \left[1 - \frac{(5 a^6 + 45 a^4 r^2 + 63 a^2 r^4 + 15 r^6) \sigma^6}{15 (a - r)^6 (a + r)^6} \right], \quad r < r_c$$

A_{cs} is the Hamaker constant, a is the radius of the colloidal particle, and R_c 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.

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)
- σ (distance units)
- $d1$ (distance units)
- $d2$ (distance units)
- cutoff (distance units)

A is the Hamaker energy prefactor and should typically be set as follows:

- $A_{cc} = \text{colloid/colloid} = 4 \pi^2 = 39.5$
- $A_{cs} = \text{colloid/solvent} = \sqrt{A_{cc} * A_{ss}}$
- $A_{ss} = \text{solvent/solvent} = 144$ (assuming $\epsilon = 1$, so that $144/36 = 4$)

σ is the size of the solvent particle or the constituent particles integrated over in the colloidal particle and should typically be set as follows:

- $\sigma_{cc} = \text{colloid/colloid} = 1.0$
- $\sigma_{cs} = \text{colloid/solvent} = \text{arithmetic mixing between colloid } \sigma \text{ and solvent } \sigma$
- $\sigma_{ss} = \text{solvent/solvent} = 1.0$ or whatever size the solvent particle is

Thus typically $\sigma_{cs} = 1.0$, unless the solvent particle's size $\neq 1.0$.

$d1$ 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 σ . 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 $I \neq J$, 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.

Related commands:

[pair_coeff](#)

Default: none

(**Everaers**) Everaers, Ejtehad, Phys Rev E, 67, 041710 (2003).

pair_style coul/cut command

pair_style coul/debye command

pair_style coul/long command

Syntax:

```
pair_style coul/cut cutoff
pair_style coul/debye kappa cutoff
pair_style coul/long 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

$$E = \frac{Cq_iq_j}{\epsilon r} \quad r < r_c$$

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

$$E = \frac{Cq_iq_j}{\epsilon r} \exp(-\kappa r) \quad r < r_c$$

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.

Mixing, shift, table, tail correction, restart, rRESPA info:

For atom type pairs I,J and $I \neq J$, 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.

On some 64-bit machines, compiling with `-O3` appears to break the Coulombic tabling option used by the *coul/long* style. See the "Additional build tips" section of the Making LAMMPS documentation pages for workarounds on this issue.

Related commands:

[pair_coeff](#), [pair_style hybrid/overlay](#)

Default: none

pair_style dipole/cut command

Syntax:

```
pair_style dipole/cut 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
```

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.

$$\begin{aligned}
 E_{qq} &= \frac{q_i q_j}{r} \\
 E_{qp} &= \frac{q}{r^3} (\vec{p} \cdot \vec{r}) \\
 E_{pp} &= \frac{1}{r^3} (\vec{p}_i \cdot \vec{p}_j) - \frac{3}{r^5} (\vec{p}_i \cdot \vec{r})(\vec{p}_j \cdot \vec{r})
 \end{aligned}$$

$$\begin{aligned}
 F_{qq} &= \frac{q_i q_j}{r^3} \vec{r} \\
 F_{qp} &= -\frac{q}{r^3} \vec{p} + \frac{3q}{r^5} (\vec{p} \cdot \vec{r}) \vec{r} \\
 F_{pp} &= \frac{3}{r^5} (\vec{p}_i \cdot \vec{p}_j) \vec{r} - \frac{15}{r^7} (\vec{p}_i \cdot \vec{r})(\vec{p}_j \cdot \vec{r}) \vec{r} + \frac{3}{r^5} [(\vec{p}_j \cdot \vec{r}) \vec{p}_i + (\vec{p}_i \cdot \vec{r}) \vec{p}_j]
 \end{aligned}$$

$$\begin{aligned}
 T_{pq} = T_{ij} &= \frac{q_j}{r^3} (\vec{p}_i \times \vec{r}) \\
 T_{qp} = T_{ji} &= -\frac{q_i}{r^3} (\vec{p}_j \times \vec{r}) \\
 T_{pp} = T_{ij} &= -\frac{1}{r^3} (\vec{p}_i \times \vec{p}_j) + \frac{3}{r^5} (\vec{p}_j \cdot \vec{r})(\vec{p}_i \times \vec{r}) \\
 T_{pp} = T_{ji} &= -\frac{1}{r^3} (\vec{p}_j \times \vec{p}_i) + \frac{3}{r^5} (\vec{p}_i \cdot \vec{r})(\vec{p}_j \times \vec{r})
 \end{aligned}$$

where q_i and q_j are the charges on the two particles, \vec{p}_i and \vec{p}_j are the dipole moment vectors of the two particles, r is their separation distance, and the vector $\vec{r} = \vec{R}_i - \vec{R}_j$ is the separation vector between the two particles. Note that

E_{qq} and F_{qq} are simply Coulombic energy and force, $F_{ij} = -F_{ji}$ as symmetric forces, and $T_{ij} \neq -T_{ji}$ since the torques do not act symmetrically. These formulas are discussed in (Allen) and in (Toukmaji).

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.

Use of this pair style requires the use of the `fix nve/dipole` command to integrate rotation of the dipole moments. Additionally, `atom_style dipole` 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 $I \neq J$, 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 $I \neq J$, 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 Lennard–Jones portion 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 "dipole" 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/dipole](#), [compute temp/dipole](#)

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).

pair_style dpd command

Syntax:

```
pair_style dpd T cutoff seed
```

- T = temperature (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
```

Description:

Style *dpd* computes a force field for dissipative particle dynamics (DPD) following the exposition in [\(Groot\)](#). The force on atom I due to atom J is given as a sum of 3 terms

$$\begin{aligned}
 \vec{f} &= (F^C + F^D + F^R)\hat{r}_{ij} & r < r_c \\
 F^C &= Aw(r) \\
 F^D &= -\gamma w^2(r)(\hat{r}_{ij} \bullet \vec{v}_{ij}) \\
 F^R &= \sigma w(r)\alpha(\Delta t)^{-1/2} \\
 w(r) &= 1 - r/r_c
 \end{aligned}$$

where F_c is a conservative force, F_d is a dissipative force, and F_r is a random force. \hat{r}_{ij} is a unit vector in the direction $\mathbf{r}_i - \mathbf{r}_j$, \mathbf{V}_{ij} is the vector difference in velocities of the two atoms = $\mathbf{V}_i - \mathbf{V}_j$, α is a Gaussian random number with zero mean and unit variance, Δt is the timestep size, and $w(r)$ is a weighting factor that varies between 0 and 1. r_c is the cutoff. σ is set equal to $\sqrt{2 T \gamma}$, where T is a parameter in the *pair_style* command.

The pairwise energy associated with this potential is only due to the conservative force term F_c .

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 σ 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.

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 "dpd" package. It is only enabled if LAMMPS was built with that package. See the [Making LAMMPS](#) section for more info.

The default frequency for rebuilding neighbor lists is every 10 steps (see the [neigh_modify](#) command). This may be too infrequent for 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.

Related commands:

[pair_coeff](#)

Default: none

(Groot) Groot and Warren, J Chem Phys, 107, 4423–35 (1997).

pair_style eam command**pair_style eam/opt command****pair_style eam/alloy command****pair_style eam/alloy/opt command****pair_style eam/fs command****pair_style eam/fs/opt command****Syntax:**

```
pair_style style
```

- style = *eam* or *eam/alloy* or *eam/fs* or *eam/opt* or *eam/alloy/opt* or *eam/fs/opt*

Examples:

```
pair_style eam
pair_style eam/opt
pair_coeff * * cuu3
pair_coeff 1*3 1*3 niu3.eam
```

```
pair_style eam/alloy
pair_style eam/alloy/opt
pair_coeff * * ../potentials/nialhjea.eam.alloy Ni Al Ni Ni
```

```
pair_style eam/fs
pair_style eam/fs/opt
pair_coeff * * nialhjea.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 E_i 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 ρ , ϕ is a pair potential interaction, and α and β 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.

Style *eam/opt* is an optimized version of style *eam* that should give identical answers. Depending on system size and the processor you are running on, it may be 5–25% faster (for the pairwise portion of the run time).

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/fs* styles as described below. These files require no mixing since they specify alloy interactions explicitly.

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.

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 $I \neq J$ 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 $F(\rho)$ (N_{ρ} values)
- effective charge function $Z(r)$ (N_r values)
- density function $\rho(r)$ (N_r 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 $r = 0, dr, 2*dr, \dots (N_r-1)*dr$.

The units for the embedding function F are eV. The units for the density function ρ are the same as for *drho* (see above, electron density). The units for the effective charge Z are "atomic charge" or $\sqrt{\text{Hartree} * \text{Bohr-radius}}$. 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

$$r*\phi = 27.2 * 0.529 * Z_i * Z_j$$

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.

Style *eam/alloy/opt* is an optimized version of style *eam/alloy* that should give identical answers. Depending on system size and the processor you are running on, it may be 5–25% faster (for the pairwise portion of the run time).

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/nialhjea.setfl` file has tabulated EAM values for 3 elements and their alloy interactions: Ni, Al, and H. If your LAMMPS simulation has 4 atom 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 * * nialhjea.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 (g/cm³) 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/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 E_i 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.

Style *eam/fs/opt* is an optimized version of style *eam/fs* that should give identical answers. Depending on system size and the processor you are running on, it may be 5–25% faster (for the pairwise portion of the run time).

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 * * nialhjea.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 ($i,j \neq j,i$) so there are N_{elements}^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 ($r \cdot \phi$, 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 $i \geq j$ are listed.

Mixing, shift, table, tail correction, restart, rRESPA info:

For atom type pairs I,J and $I \neq J$, 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 $I \neq J$ 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 those ending in *opt* are part of the "manybody" package. They are only enabled if LAMMPS was built with that package (which it is by default). The styles ending in *opt* are part of the "opt" package and also require the "manybody" package. They are 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).

pair_style gayberne command

Syntax:

```
pair_style gayberne gamma epsilon mu cutoff
```

- gamma = shift for potential minimum (typically 1)
- epsilon = 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:

Style *gayberne* computes a Gay–Berne anisotropic LJ interaction ([Berardi](#)) between pairs of ellipsoidal particles or an ellipsoidal and spherical particle via the formulas

$$U(\mathbf{A}_1, \mathbf{A}_2, \mathbf{r}_{12}) = U_r(\mathbf{A}_1, \mathbf{A}_2, \mathbf{r}_{12}, \gamma) \cdot \eta_{12}(\mathbf{A}_1, \mathbf{A}_2, v) \cdot \chi_{12}(\mathbf{A}_1, \mathbf{A}_2, \mathbf{r}_{12}, \mu)$$

$$U_r = 4\epsilon(\varrho^{12} - \varrho^6)$$

$$\varrho = \frac{\sigma}{h_{12} + \gamma\sigma}$$

where \mathbf{A}_1 and \mathbf{A}_2 are the transformation matrices from the simulation box frame to the body frame and \mathbf{r}_{12} is the center to center vector between the particles. U_r controls the shifted distance dependent interaction based on the distance of closest approach of the two particles (h_{12}) and the user-specified shift parameter γ . 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 its RE-squared potential fix. Details on the epsilon and mu parameters are given [here](#). 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 of the ellipsoidal particles.

More details of the Gay–Berne formulation are given in the references listed below and in [this supplementary document](#).

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.

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 are all 1.0, and if the shape of the particle is spherical (see the [shape](#) command), meaning the 3 diameters are all the same, then the particle is treated as "spherical" by the Gay–Berne potential. This is significant because if two "spherical" particles interact, then the simple Lennard–Jones formula is used to compute their interaction energy/force using epsilon and sigma, which is much cheaper to compute than the full Gay–Berne formula. Thus you should insure epsilon a,b,c are set to 1.0 for spherical particle types and use epsilon and sigma to specify its interaction with other spherical particles.

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:

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.

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/hertzian command

pair_style gran/history command

pair_style gran/no_history command

Syntax:

```
pair_style style Kn gamma_n xmu dampflag
```

- style = *gran/hertzian* or *gran/history* or *gran/no_history*
- Kn = spring constant for particle repulsion (force/distance units)
- gamma_n = damping coefficient for normal direction collisions (1/time units)
- xmu = static yield criterion (unitless)
- dampflag = 0 or 1 if tangential damping force is excluded or included

Examples:

```
pair_style gran/history 200000.0 0.5 1.0 1
```

Description:

The *gran* styles use the following formula ([Silbert](#)) for frictional force between two granular particles that are a distance r apart when r is less than the contact distance $d = R_i + R_j$, where R_i and R_j are the radii of the two particles:

$$\mathbf{F} = f(\delta/d) (k_n \delta \mathbf{n}_{ij} - m_{\text{eff}} \gamma_n \mathbf{v}_n) + f(\delta/d) (-k_t \delta \Delta \mathbf{s}_t - m_{\text{eff}} \gamma_t \mathbf{v}_t)$$

The 1st term is a normal force and the 2nd term is a tangential force. The normal force has 2 parts: a contact force and a damping force. The tangential force also has 2 parts: a shear force and a damping force. The shear force is included in pair styles *history* and *Hertzian*, but is not included in pair style *no_history*. The tangential damping force is not included if *dampflag* is set to 0. The other quantities in the equation are as follows:

- $\delta = d - r$
- $f(x) = 1$ for Hookean contact used in pair styles *history* and *no_history*
- $f(x) = \sqrt{x}$ for Hertzian contact used in pair style *hertzian*
- K_n = elastic constant for normal contact
- K_t = elastic constant for tangential contact = $2/7$ of K_n
- γ_n = viscoelastic damping constant for normal contact
- γ_t = viscoelastic damping constant for tangential contact = $1/2$ of γ_n
- $m_{\text{eff}} = M_i M_j / (M_i + M_j)$ = effective mass of 2 particles of mass M_i and M_j
- $\Delta \mathbf{s}_t$ = tangential displacement vector between the 2 spherical particles which is truncated to satisfy a frictional yield criterion
- \mathbf{n}_{ij} = unit vector along the line connecting the centers of the 2 particles
- \mathbf{v}_n = normal component of the relative velocity of the 2 particles
- \mathbf{v}_t = tangential component of the relative velocity of the 2 particles

The `Kn` and `gamma_n` coefficients are set as parameters to the `pair_style` command. You can also think of `Kn` as being in `mg/d` units where `m` is mass, `g` is the gravitational constant, and `d` is the characteristic diameter of a particle.

`Xmu` is also specified in the `pair_style` command and is the upper limit of the tangential force through the Coulomb criterion $F_t = xmu * F_n$. The tangential force between 2 particles grows according to a tangential spring and dash-pot model until $F_t/F_n = xmu$ and is then held at $F_t = F_n * xmu$ until the particles lose contact.

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. If a granular potential is used as part 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.

See the citation below for more discussion of granular potentials.

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.

Related commands:

[pair_coeff](#)

Default: none

(**Silbert**) Silbert, Ertas, Grest, Halsey, Levine, Plimpton, Phys Rev E, 64, p 051302 (2001).

pair_style lj/gromacs command

pair_style lj/gromacs/coul/gromacs 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](#).

$$\begin{aligned}
 E_{LJ} &= 4\epsilon \left[\left(\frac{\sigma}{r} \right)^{12} - \left(\frac{\sigma}{r} \right)^6 \right] + S_{LJ}(r) & r < r_c \\
 E_C &= \frac{Cq_i q_j}{\epsilon r} + S_C(r) & r < r_c \\
 S(r) &= 0 & r < r_1 \\
 S(r) &= A(r - r_1)^2 + B(r - r_1)^3 & r_1 < r < r_c
 \end{aligned}$$

R_1 is the inner cutoff; R_c 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: $S(r_1) = S'(r_1) = 0$, $S(r_c) = -F(r_c)$, $S'(r_c) = -F'(r_c)$, where $F(r)$ is the corresponding 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.

Mixing, shift, table, tail correction, restart, rRESPA info:

For atom type pairs I,J and $I \neq J$, 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 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.

The *pair_coeff* commands are also specified exactly as they would be for a simulation using only one pair style, with one additional argument. Following the I,J type specification, the first argument sets the pair sub-style. The remaining arguments are the coefficients 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
```

Note that the `pair_coeff` command for *eam/alloy* includes a mapping specification of elements to all atom types, even those not assigned to the *eam/alloy* potential. The NULL keyword is used by 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. Thus if you specify the same atom type pair in a second `pair_coeff` command, a second sub-style is added to the list of potentials that will be calculated for two interacting atoms of those types.

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:

For both the *hybrid* and *hybrid/overlay* styles, every atom type pair I,J (where $I \leq J$) 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:

```
pair_coeff      2 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.

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 $I \neq J$, 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 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 be 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/opt command****pair_style lj/cut/coul/cut command****pair_style lj/cut/coul/debye command****pair_style lj/cut/coul/long command****pair_style lj/cut/coul/long/tip4p command****Syntax:**

```
pair_style style args
```

- style = *lj/cut* or *lj/cut/opt* 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/opt 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_style lj/cut/opt 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

$$E = 4\epsilon \left[\left(\frac{\sigma}{r} \right)^{12} - \left(\frac{\sigma}{r} \right)^6 \right] \quad r < r_c$$

Rc is the cutoff.

Style *lj/cut/opt* is an optimized version of style *lj/cut* that should give identical answers. Depending on system size and the processor you are running on, it may be 5–25% faster (for the pairwise portion of the run time).

Style *lj/cut/coul/cut* adds a Coulombic pairwise interaction given by

$$E = \frac{Cq_iq_j}{\epsilon r} \quad r < r_c$$

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

$$E = \frac{Cq_iq_j}{\epsilon r} \exp(-\kappa r) \quad r < r_c$$

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.

Mixing, shift, table, tail correction, restart, rRESPA info:

For atom type pairs I,J and $I \neq J$, 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 of 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 "k-space" package. They are only enabled if LAMMPS was built with that package (which it is by default). The *lj/cut/opt* style is part of the "opt" package. It is only enabled if LAMMPS was built with that package. See the [Making LAMMPS](#) section for more info.

On some 64-bit machines, compiling with `-O3` appears to break the Coulombic tabling option used by the *lj/cut/coul/long* style. See the "Additional build tips" section of the Making LAMMPS documentation pages for workarounds on this issue.

Related commands:

[pair_coeff](#)

Default: none

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

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

$$E = 4\epsilon \left[\left(\frac{\sigma}{r} \right)^{12} - \left(\frac{\sigma}{r} \right)^6 \right] \quad r < r_c$$

$$E = \frac{Cq_iq_j}{\epsilon r} \quad r < r_c$$

where C is an energy–conversion constant, Q_i and Q_j are the charges on the 2 atoms, ϵ is the dielectric constant which can be set by the [dielectric](#) command, and R_c 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 $I \neq J$, 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 styles supports the [pair_modify](#) tail option for adding a long-range tail correction to the Lennard-Jones portion of the energy and pressure of the pair interaction, assuming *flag_lj* is *cut*.

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.

On some 64-bit machines, compiling with `-O3` appears to break the Coulombic tabling option used by the *lj/coul* style. See the "Additional build tips" section of the Making LAMMPS documentation pages for workarounds on this issue.

Related commands:

[pair_coeff](#)

Default: none

(In 't Veld) In 't Veld, Ismail, Grest, J Chem Phys (accepted) (2007).

pair_style lj/expand 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:

$$E = 4\epsilon \left[\left(\frac{\sigma}{r - \Delta} \right)^{12} - \left(\frac{\sigma}{r - \Delta} \right)^6 \right] \quad r < r_c + \Delta$$

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.

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/smooth 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.

$$\begin{aligned} E &= 4\epsilon \left[\left(\frac{\sigma}{r} \right)^{12} - \left(\frac{\sigma}{r} \right)^6 \right] & r < r_{in} \\ F &= C_1 + C_2(r - r_{in}) + C_3(r - r_{in})^2 + C_4(r - r_{in})^3 & r_{in} < r < r_c \end{aligned}$$

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)
- inner (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.

Mixing, shift, table, tail correction, restart, rRESPA info:

For atom type pairs I, J and $I \neq J$, 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 = viscosity (mass/distance/time 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 * *
```

Description:

Style *lubricate* computes pairwise interactions between mono-disperse spherical particles via this formula from [\(Ball and Melrose\)](#)

$$W = \frac{-a_{sq} |(v_1 - v_2) \bullet \mathbf{nn}|^2 - a_{sh} |(\omega_1 + \omega_2) \bullet (\mathbf{I} - \mathbf{nn}) - 2\Omega_N|^2 - a_{pw} |(\omega_1 - \omega_2) \bullet (\mathbf{I} - \mathbf{nn})|^2 - a_{tw} |(\omega_1 - \omega_2) \bullet \mathbf{nn}|^2}{r} \quad r < r_c$$

$$\Omega_N = \mathbf{n} \times (v_1 - v_2)/r$$

which represents the dissipation W between two nearby particles due to their relative velocities in the presence of a background solvent. R_c is the outer cutoff specified in the *pair_style* command, the translational velocities of the 2 particles are v_1 and v_2 , the angular velocities are w_1 and w_2 , and \mathbf{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. 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 $r_{ij} - 2*\text{radius}$, 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 thermostating 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 $I \neq J$, 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.

Because this potential computes forces and torques on particles, the atom style must support particles whose size is set via the [shape](#) command. This is [atom_style](#) ellipsoid and dipole. Since only spherical mono-disperse particles are currently allowed for [pair_style lubricate](#), this means the 3 shape radii for all particle types must be the same.

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:

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 ρ , and ϕ 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.

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
              bl  = rock salt (NaCl structure)
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
augtl        = 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

```

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.

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](#).

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 $I \neq J$ 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 is is not necessary to specify coefficients when $I \neq J$, 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 $I \neq J$ 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/opt command

Syntax:

```
pair_style morse cutoff
```

- cutoff = global cutoff for Morse interactions (distance units)

Examples:

```
pair_style morse 2.5
pair_style morse/opt 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

$$E = D_0 \left[e^{-2\alpha(r-r_0)} - 2e^{-\alpha(r-r_0)} \right] \quad r < r_c$$

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.

Style *morse/opt* is an optimized version of style *morse* that should give identical answers. Depending on system size and the processor you are running on, it may be 5–25% faster (for the pairwise portion of the run time).

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:

The *morse/opt* style is part of the "opt" 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

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

Syntax:

```
pair_style peri/pmb
```

Examples:

```
pair_style peri/pmb
pair_coeff * * 1.6863e22 0.0015001 0.0005 0.25
```

Description:

Style *peri/pmb* style implements the Peridynamic bond-based prototype microelastic brittle (PMB) model, which can be used to model materials at the mesoscopic or macroscopic scale. The canonical paper on Peridynamics is ([Silling](#)). The implementation of Peridynamics in LAMMPS is described in ([Parks](#)). Also see the [PDLAMMPS user guide](#) for more details about this particular potential and using it 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:

- *c* (energy/distance/volume² 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 $c/\text{distance} = \text{stiffness}/\text{volume}^2$, where stiffness is energy/distance² and volume is distance³. See the users guide for more details.

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.

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:

The *peri/pmb* style 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:

[pair_coeff](#)

Default: none

(Parks) Parks, Lehoucq, Plimpton, Silling, to appear in Comp Phys Comm, (2008).

(Silling) Silling, J Mech Phys Solids, 48, 175–209 (2000).

pair_style resquared 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 of the ellipsoidal particles and the [shape](#) command should be used to specify ellipsoid diameters.

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. ellipsoid or LJ sphere. The type of particle is determined by the diameters specified with the [shape](#) command. LJ spheres have diameters equal to zero and thus represent a single particle with size sigma. The epsilon_i_* or epsilon_j_* parameters are ignored for LJ sphere interactions. The interactions between two LJ sphere particles are computed using the standard Lennard–Jones formula.

A12 specifies the energy prefactor which depends on the type of particles interacting. For ellipsoid–ellipsoid interactions, A12 is the Hamaker constant as described in ([Everaers](#)). In LJ units:

$$A_{12} = 4\pi^2 \epsilon_{LJ} (\rho\sigma^3)^2$$

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, A12 gives the energy prefactor (see [here](#) for details):

$$A_{12} = 4\pi^2 \epsilon_{\text{LJ}} (\rho \sigma^3)$$

For LJ sphere-LJ sphere interactions, A12 is the standard epsilon used in Lennard-Jones pair styles:

$$A_{12} = \epsilon_{\text{LJ}}$$

sigma specifies the diameter of the continuous distribution of constituent particles within each ellipsoid used to model the RE-squared potential. Therefore, the effective shape of an ellipsoid is given by the specified diameters (see the [shape](#) command) plus sigma.

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.

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 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.

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 LJ 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-LJ 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 style 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.

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 * * 0.0 60.0
pair_coeff 1 1 0.0 60.0 3.0
```

Description:

Style *soft* computes pairwise interactions with the formula

$$E = A \left[1 + \cos \left(\frac{\pi r}{r_c} \right) \right] \quad r < r_c$$

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 varies in time from the start to the end of the run. The [run](#) command documents how to make the ramping take place across multiple runs. R_c 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:

- Astart (energy units)
- Astop (energy units)
- cutoff (distance units)

Astart and Astop are the values of the prefactor at the start and end of the next run. At intermediate times the value of A will be ramped between these 2 values. Note that before performing a 2nd run, you will want to adjust the values of Astart and Astop for all type pairs, or switch to a new pair style.

The last coefficient is optional. If not specified, the global soft cutoff is used.

Mixing, shift, table, tail correction, restart, rRESPA info:

For atom type pairs I, J and $I \neq J$, the Astart, Astop coefficients and cutoff distance for this pair style can be mixed. Astart and Astop are 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](#)

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 airebo](#) – AI-REBO potential
- [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 eam](#) – embedded atom method (EAM)
- [pair_style eam/opt](#) – optimized version of EAM
- [pair_style eam/alloy](#) – alloy EAM
- [pair_style eam/alloy/opt](#) – optimized version of alloy EAM
- [pair_style eam/fs](#) – Finnis–Sinclair EAM
- [pair_style eam/fs/opt](#) – optimized version of Finnis–Sinclair EAM
- [pair_style gayberne](#) – Gay–Berne ellipsoidal potential
- [pair_style gran/hertzian](#) – granular potential with Hertzian interactions
- [pair_style gran/history](#) – granular potential with history effects
- [pair_style gran/no_history](#) – granular potential without history effects
- [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/charmm/coul/long/opt](#) – optimized version of 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/opt](#) – optimized version of cutoff LJ
- [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 lubricate](#) – hydrodynamic lubrication forces
- [pair_style meam](#) – modified embedded atom method (MEAM)
- [pair_style morse](#) – Morse potential
- [pair_style morse/opt](#) – optimized version of Morse potential
- [pair_style peri/pmb](#) – peridynamic PMB potential
- [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

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](#).

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 * * SiC.sw Si C Si
```

Description:

The *sw* style computes a 3-body [Stillinger–Weber](#) potential for the energy E of a system of atoms as

$$E = \sum_i \sum_{j>i} \phi_2(r_{ij}) + \sum_i \sum_{j \neq i} \sum_{k>j} \phi_3(r_{ij}, r_{ik}, \theta_{ijk})$$

$$\phi_2(r_{ij}) = A_{ij} \epsilon_{ij} \left[B_{ij} \left(\frac{\sigma_{ij}}{r_{ij}} \right)^{p_{ij}} - \left(\frac{\sigma_{ij}}{r_{ij}} \right)^{q_{ij}} \right] \exp \left(\frac{\sigma_{ij}}{r_{ij} - a_{ij} \sigma_{ij}} \right)$$

$$\phi_3(r_{ij}, r_{ik}, \theta_{ijk}) = \lambda_{ijk} \epsilon_{ijk} [\cos \theta_{ijk} - \cos \theta_{0ijk}]^2 \exp \left(\frac{\gamma_{ij} \sigma_{ij}}{r_{ij} - a_{ij} \sigma_{ij}} \right) \exp \left(\frac{\gamma_{ik} \sigma_{ik}}{r_{ik} - a_{ik} \sigma_{ik}} \right)$$

where ϕ_2 is a two-body term and ϕ_3 is a three-body term. The summations in the formula are over all neighbors J and K of atom I within a cutoff distance = $a \cdot \sigma$.

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 the `SiC.sw` file has Stillinger–Weber values for Si and C. If your LAMMPS simulation has 4 atom 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.

Mixing, shift, table, tail correction, restart, rRESPA info:

For atom type pairs I, J and $I \neq J$, 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 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 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. The pair distance is used to index into the table via a fast bit-mapping technique ([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, or in the data file or restart files read by the [read_data](#) or [read_restart](#) commands:

- 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 as large an inner cutoff as possible. This avoids fitting splines to very steep parts of the potential.
 - Make sure the tabulated potential you are feeding to LAMMPS is not pathological.
-

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; its value is the number of table entries that follow. 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 be 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

$$\begin{aligned}
 E &= \frac{1}{2} \sum_i \sum_{j \neq i} V_{ij} \\
 V_{ij} &= f_C(r_{ij}) [f_R(r_{ij}) + b_{ij} f_A(r_{ij})] \\
 f_C(r) &= \begin{cases} 1 & ; r < R - D \\ \frac{1}{2} - \frac{1}{2} \sin\left(\frac{\pi}{2} \frac{r-R}{D}\right) & ; R - D < r < R + D \\ 0 & ; r > R + D \end{cases} \\
 f_R(r) &= A \exp(-\lambda_1 r) \\
 f_A(r) &= -B \exp(-\lambda_2 r) \\
 b_{ij} &= (1 + \beta^n \zeta_{ij}^n)^{-\frac{1}{2n}} \\
 \zeta_{ij} &= \sum_{k \neq i, j} f_C(r_{ik}) g(\theta_{ijk}) \exp[\lambda_3^m (r_{ij} - r_{ik})^m] \\
 g(\theta) &= \gamma_{ijk} \left(1 + \frac{c^2}{d^2} - \frac{c^2}{[d^2 + (\cos \theta - \cos \theta_0)^2]} \right)
 \end{aligned}$$

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. 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 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 $\lambda_3 = 0$ 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.

Mixing, shift, table, tail correction, restart, rRESPA info:

For atom type pairs I, J and $I \neq J$, 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)

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

$$\begin{aligned}
 E &= \frac{1}{2} \sum_i \sum_{j \neq i} V_{ij} \\
 V_{ij} &= (1 - f_F(r_{ij})) V_{ij}^{ZBL} + f_F(r_{ij}) V_{ij}^{Tersoff} \\
 f_F(r_{ij}) &= \frac{1}{1 + e^{-A_F(r_{ij} - r_C)}} \\
 V_{ij}^{ZBL} &= \frac{1}{4\pi\epsilon_0} \frac{Z_1 Z_2 e^2}{r_{ij}} \phi(r_{ij}/a) \\
 a &= \frac{0.8854 a_0}{Z_1^{0.23} + Z_2^{0.23}} \\
 \phi(x) &= 0.1818e^{-3.2x} + 0.5099e^{-0.9423x} + 0.2802e^{-0.4029x} + 0.02817e^{-0.2016x} \\
 V_{ij}^{Tersoff} &= f_C(r_{ij}) [f_R(r_{ij}) + b_{ij} f_A(r_{ij})] \\
 f_C(r) &= \begin{cases} 1 & : r < R - D \\ \frac{1}{2} - \frac{1}{2} \sin\left(\frac{\pi}{2} \frac{r - R}{D}\right) & : R - D < r < R + D \\ 0 & : r > R + D \end{cases} \\
 f_R(r) &= A \exp(-\lambda_1 r) \\
 f_A(r) &= -B \exp(-\lambda_2 r) \\
 b_{ij} &= (1 + \beta^n \zeta_{ij}^n)^{-\frac{1}{2n}} \\
 \zeta_{ij} &= \sum_{k \neq i,j} f_C(r_{ik}) g(\theta_{ijk}) \exp[\lambda_3^3 (r_{ij} - r_{ik})^m] \\
 g(\theta) &= \gamma_{ijk} \left(1 + \frac{c^2}{d^2} - \frac{c^2}{[d^2 + (\cos \theta - \cos \theta_0)^2]} \right)
 \end{aligned}$$

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 ϵ_0 as the permittivity of vacuum. The second part is the ZBL universal screening function, with a_0 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
- γ
- λ_3 (1/distance units)
- c
- d
- $\cos\theta_0$ (can be a value < -1 or > 1)
- n
- β
- λ_2 (1/distance units)
- B (energy units)
- R (distance units)
- D (distance units)
- λ_1 (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 $I \neq J$, 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. Erhart and 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 $N \geq 12$, an *inner* value that is smaller than the distance of closest approach of 2 atoms, and an *outer* value \leq 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 $Q_i = Q_j = 1.0$ 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

$$E = A \frac{e^{-\kappa r}}{r} \quad r < r_c$$

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

print command

Syntax:

```
print string
```

- string = text string to print, which may contain variables

Examples:

```
print "Done with equilibration"  
print "The system volume is now $v"
```

Description:

Print a text string to the screen and logfile. The text string must be a single argument, so it should be enclosed in double quotes if it is more than one word. 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
```

Description:

Specify how processors are mapped as a 3d logical grid to the global simulation box.

When this command has not been specified, LAMMPS will choose Px, Py, Pz based on the dimensions of the global simulation box so as to minimize the surface/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 should 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 over the course of the simulation.

The product of Px, Py, Pz must equal P, the total # of processors LAMMPS is running on. 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.

If P is large and prime, 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:

LAMMPS chooses 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
- *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 domain

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. See the [region prism](#) command for a description of how the extent of the parallelepiped is defined. The parallelepiped has its "origin" at (xlo,ylo,zlo) and 3 edge vectors starting from the origin given by $a = (xhi-xlo,0,0)$; $b = (xy,yhi-ylo,0)$; $c = (xz,yz,zhi-zlo)$. Note that if your simulation will tilt the box, e.g. via the [fix deform](#) command, the simulation box must be triclinic, even if the tilt factors are initially 0.0.

The tilt factors (xy,xz,yz) can 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 equivalent.

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.

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; 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 and may cause a parallel simulation to lose atoms if 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.

These are the section keywords for the body of the file.

- *Atoms, Velocities, Masses, Shapes, Dipoles* = atom-property sections
- *Bonds, Angles, Dihedrals, Improvers* = 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
dpd	atom-ID atom-type x y z
ellipsoid	atom-ID atom-type x y z quatw quati quatj quatk
full	atom-ID molecule-ID atom-type q x y z
granular	atom-ID atom-type diameter density x y z
molecular	atom-ID molecule-ID atom-type x y z
peri	atom-ID atom-type volume density 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
- type-ID = type of atom (1-Ntype)
- q = charge on atom (charge units)
- diameter = diameter of atom (distance units)
- density = density of atom (mass/distance³ units)
- volume = volume of atom (distance³ units)
- x,y,z = coordinates of atom
- mux,muy,muz = direction of dipole moment of atom
- quatw,quati,quatj,quatk = quaternion components for orientation of atom

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), 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 values *quatw*, *quati*, *quatj*, and *quatk* set the orientation of the atom as a quaternion (4-vector). Note that the [shape](#) command or "Shapes" section of the data file specifies 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*) was not a unit vector.

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 granular
```

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).

Dipoles section:

- one line per atom type line syntax: ID dipole-moment

```
ID = atom type (1-N)
dipole-moment = value of dipole moment
```

- example:

```
2 0.5
```

This defines the dipole moment of each atom type (which can be 0.0 for some types). This can also be set via the [dipole](#) command in the input script.

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:

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-Nimproptype)
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 should not be used for atom styles that define a mass for individual atoms – e.g. atom style granular.

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.

Shapes section:

- one line per atom type
- line syntax: ID x y z

```
ID = atom type (1-N)
x = x diameter
y = y diameter
z = z diameter
```

- example:

```
3 2.0 1.0 1.0
```

This defines the shape of each atom type. This can also be set via the [shape](#) command in the input script. This section should only be used for atom styles that define a shape, e.g. atom style dipole or ellipsoid.

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
ellipsoid	atom-ID vx vy vz lx ly lz
granular	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 granular atom

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 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. Note that only a single processor reads all the files, so the input does not use parallel I/O. The number of processors which created the set can be different the number of processors in the current LAMMPS simulation.

A restart file stores the following information about a simulation: units and atom style, simulation box size and shape and boundary settings, group definitions, atom type settings such as mass and particle shape, individual atoms and 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 `kpace_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` and `dump` output, `geometric regions`, etc.

Restrictions: none

Related commands:

`read_data`, `write_restart`, `restart`

Default: none

region command

Syntax:

```
region ID style args keyword value ...
```

- ID = user-assigned name for the region
- style = *block* or *cylinder* or *prism* or *sphere* or *union* or *intersect*

```
block args = xlo xhi ylo yhi zlo zhi
             xlo,xhi,ylo,yhi,zlo,zhi = bounds of block in all
             dimensions (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)
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/value pairs may be appended
- keyword = *side* or *units*

```
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
```

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 INF units box
region 1 prism 0 10 0 10 0 10 2 0 0
region outside union 4 side1 side2 side3 side4
```

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 the atoms in the region can be identified as a group via the [group](#) command, or deleted via the [delete_atoms](#) command.

The lo/hi values for *block* 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 style *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 *prism*, a parallelepiped is defined (it's too hard to spell parallelepiped in an input script!). Think of the parallelepiped as initially an axis-aligned orthogonal box with the same xyz lo/hi parameters as region style *block* would define. Then, while holding the (xlo,ylo,zlo) corner point fixed, the box is "skewed" or "tilted" in 3 directions. First, for the lower xy face of the box, the xy factor is how far the upper y edge is shifted in the x direction. The lower xy face is now a parallelogram. A plus or minus value for xy can be specified; 0.0 means no tilt. Then, the upper xy face of the box is translated in the x and y directions by xz and yz. This results in a parallelepiped whose "origin" is at (xlo,ylo,zlo) with 3 edge vectors starting from its origin given by $a = (xhi-xlo,0,0)$; $b = (xy,yhi-ylo,0)$; $c = (xz,yz,zhi-zlo)$.

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 equivalent.

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. 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.

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 and units = lattice.

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 time or the timestep in order to perform time-dependent operations. Examples include the "ave" or "wall" fixes such as [fix ave/spatial](#) or [fix wall/lj126](#), and also [fix indent](#). The wall and indeter fixes allow for a velocity or other time-dependent parameter to be specified, which would be messed up by resetting the timestep.

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 on parallel machines that support parallel I/O for output.

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 command
  M = break the run into M-timestep segments and invoke a command between them
  command = a single LAMMPS command listed the same as if on a line by itself
  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. The [pair_style soft](#) potential also changes its pair potential coefficients in this manner.

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* option provides a means of breaking a LAMMPS run into a series of shorter runs. Optionally a single LAMMPS command can be executed in between the short runs. This is a means to avoid listing a long series of runs and 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 it 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 command is invoked. If the command is specified as NULL, 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, as in this example, the command can contain [variables](#) which will be evaluated each time the command is invoked.

IMPORTANT NOTE: For the *every* option, the command should be listed exactly as it would be if it appeared on a line by itself. Thus all remaining arguments after the M value are considered part of the LAMMPS command

(e.g. `print "Protein Rg = $r"` as in the example above). This means that, if specified, the *every* option must be the last keyword used.

IMPORTANT NOTE: For the *every* option, if the command includes a variable (e.g. `$x` or `${abc}`), and you want the variable to be evaluated afresh each time the command is invoked, then you should enclose that command argument in double quotes, as in the `"Protein Rg = $r"` example above. If you don't do this, then the variable will be substituted for only once initially when the run command is parsed, just as occurs for any other command containing a variable.

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: none

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: none

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 *group* or *region*
- ID = atom ID 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 *vx* or *vy* or *vz* or *charge* or *dipole* or *dipole/random* or *quat/random* or *diameter* 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)
vx,vy,vz value = velocity component (velocity units)
charge value = atomic charge (charge units)
dipole values = x y z
x,y,z = orientation of dipole moment vector
dipole/random value = seed
seed = random # seed (positive integer) for dipole moment orientations
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 = particle diameter (distance units)
density value = particle density (mass/distance^3 units)
volume value = particle volume (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 atom 100 x 0.5 vx 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 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 a single atom. The style *group* selects the entire group of atoms. The style *region* selects all atoms in the geometric region. The associated ID for each of these styles is either the unique atom ID (typically a number from 1 to N = the number of atoms in the simulation), the group ID, or the region ID. 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*, *vx*, *vy*, *vz*, and *charge* set the coordinates, velocity, 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 for each atom is set by the [dipole](#) command.

Keyword *dipole/random* randomizes the orientation of the dipole moment vectors of the selected atoms. The magnitude of the dipole moment for each atom is set by the [dipole](#) command. 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 the [shape](#) command is 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.

For the *dipole* and *quat* keywords, the [atom style](#) being used must support the use of dipoles or quaternions.

Keyword *diameter* sets the size of all selected particles.

Keyword *density* sets the density of all selected particles, which combined with its density determines their mass.

Keyword *volume* sets the effective size of all selected particles.

Keyword *image* sets which image of the simulation box the atom is considered to be in. It is only applied to periodic dimensions. 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.

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 [fix 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.

For the *diameter* and *density* and *volume* keywords, the [atom style](#) being used must support the use of those parameters. For example, granular particles store a diameter and density. Peridynamic particles store a volume and density.

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

shape command

Syntax:

```
shape I x y z
```

- I = atom type (see asterisk form below)
- x = x diameter
- y = y diameter
- z = z diameter

Examples:

```
shape 1 1.0 1.0 1.0
shape * 3.0 1.0 1.0
shape 2* 3.0 1.0 1.0
```

Description:

Set the shape for all atoms of one or more atom types. Shape values can also be set in the [read_data](#) data file. See the [units](#) command for what distance units to use.

Currently, only [atom_style dipole](#) and [atom_style ellipsoid](#) require that shapes be set.

Dipoles use the atom shape to compute a moment of inertia for rotational energy. Only the 1st component of the shape is used since the particles are assumed to be spherical. The value of the first component should be the same as the Lennard–Jones sigma value defined in the dipole pair potential, i.e. in [pair_style dipole](#).

Ellipsoids use the atom shape to compute a generalized inertia tensor. For example, a shape setting of 3.0 1.0 1.0 defines a particle 3x longer in x than in y or z and with a circular cross-section in yz. Ellipsoids that are spherical can be defined by setting all 3 shape components the same.

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 shape 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 specifies shape uses the same format as the arguments of the shape command in an input script, except that no wild-card asterisk can be used. For example, under the "Shapes" section of a data file, the line that corresponds to the 1st example above would be listed as

```
1 1.0 1.0 1.0
```

Restrictions:

This command must come after the simulation box is defined by a [read_data](#), [read_restart](#), or [create_box](#) command.

All shapes must be defined before a simulation is run (if the atom style requires shapes be set).

Related commands: none

Default: none

shell command

Syntax:

```
shell style args
```

- style = *cd* or *mkdir* or *mv* or *rm* or *rmdir*

```
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
```

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
```

Description:

Execute a shell command. Only a few simple file-based shell commands are supported, in Unix-style syntax. With the exception of *cd*, all commands are executed by only a single processor, so that files/directories are not being manipulated by multiple processors.

The *cd* style 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* style executes the Unix "mkdir" command to create one or more directories.

The *mv* style executes the Unix "mv" command to rename a file and/or move it to a new directory.

The *rm* style executes the Unix "rm" command to remove one or more files.

The *rmdir* style executes the Unix "rmdir" command to remove one or more directories. A directory must be empty to be successfully removed.

Restrictions:

LAMMPS does not detect errors or print warnings when any of these Unix commands execute. E.g. if the specified directory does not exist, executing the *cd* command will silently not do anything.

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 *fene* or *lj/coul* or *lj* or *coul* or *dihedral* or *extra*

```
amber values = none
charmm 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
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 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 from atom pairs that are also bonded to each other directly or indirectly. For Lennard–Jones (LJ) and Coulombic pairwise interactions, these coefficients come in sets of three. The 1st coefficient is the weighting factor on 1–2 atom pairs, which are those 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.

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 defined 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 *dihedral* keyword is 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.

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.

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](#).

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 *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 Coulombic weighting coefficients = 0.0, dihedral = no, and extra = 0.

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 (replica exchange) simulation of multiple ensembles of a system on multiple partitions of processors. The processor partitions are defined using the `-partition` command-line switch (see [this section](#)). Each ensemble's temperature is typically controlled at a different value by a fix with ID *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
```

As the tempering simulation runs for *N* timesteps, a 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.

The last argument *index* 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

```
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: none

Related commands:

[variable](#)

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.

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:1
  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
```

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 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.

The *lost* keyword determines whether LAMMPS checks for lost atoms each time it computes thermodynamics and what it does if atoms are lost. 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. 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 (e.g. format 5 %10.4g) sets the format of the Mth value printed in each output line, 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.

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.

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.

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, atoms, cpu, 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,
                    pxx, pyy, pzz, pxy, pxz, pyz
                    c_ID, c_ID[n], f_ID, f_ID[n], v_name

step = timestep
atoms = # of atoms
cpu = elapsed CPU time
temp = temperature
press = pressure
pe = total potential energy
ke = kinetic energy
etotal = total energy (pe + ke)
enthalpy = enthalpy (pe + 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
pxx,pyy,pzz,pxy,pxz,pyz = 6 components of pressure tensor
c_ID = global scalar value calculated by a compute with ID
c_ID[N] = Nth component of global vector calculated by a compute with ID
f_ID = global scalar value calculated by a fix with ID
f_ID[N] = Nth component of global vector calculated by a fix with ID
v_name = global 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, meaning 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 the "thermo_temp" 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 the 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 the "thermo_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 *c_ID* and *c_ID[N]* keywords allow global scalar or vector quantities calculated by a compute to be output. The ID in the keyword should be replaced by the actual ID of the compute that has been defined elsewhere in the input script. See the [compute](#) command for details. Note that only global scalar or vector quantities calculated by a compute can be output as thermodynamic data; per-atom quantities calculated by a compute can be output by the [dump custom](#) command. There is a [compute reduce](#) command which can sum per-atom quantities into a global scalar or vector which can be output by *thermo_style custom*.

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 as is by *thermo_style custom*. Extensive quantities may be normalized when output by the total number of atoms in the simulation (NOT the number of atoms in the compute group) depending on the [thermo_modify norm](#) option being used.

If *c_ID* is used as a keyword, then the scalar quantity calculated by the compute is printed. If *c_ID[N]* is used, then N must be an index from 1–M where M is the length of the vector calculated by the compute. See the doc pages for individual compute styles for info on what these quantities are.

The *f_ID* and *f_ID[N]* keywords allow global scalar or vector quantities calculated by a fix to be output. The ID in the keyword should be replaced by the actual ID of the fix that has been defined elsewhere in the input script. See the doc pages for individual [fix commands](#) for details of which fixes generate global values. One particularly useful fix to use in this context is the [fix ave/time](#) command, which calculates time-averages of global scalar and vector quantities calculated by other [computes](#), [fixes](#), or [variables](#).

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 as is by *thermo_style custom*. Extensive quantities may be normalized when output by the total number of atoms in the simulation (NOT the number of atoms in the fix group) depending on the [thermo_modify norm](#) option being used.

If *f_ID* is used as a keyword, then the scalar quantity calculated by the fix is printed. If *f_ID[N]* is used, then N must be an index from 1–M where M is the length of the vector calculated by the fix. See the doc pages for individual fix styles for info on which fixes calculate these global quantities and what they are. For fixes that compute a contribution to the potential energy of the system, the scalar quantity referenced by *f_ID* is typically that quantity.

The *v_name* keyword allow the current value of a variable to be output. The name in the keyword should be replaced by the actual name of the variable 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 individual 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 could then be output with these keywords as part of thermodynamic information.

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*

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 σ ε 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 $f^* = f \sigma / \epsilon$
- temperature = reduced LJ temperature, where $T^* = T K_b / \epsilon$
- pressure = reduced LJ pressure, where $P^* = P \sigma^3 / \epsilon$
- viscosity = reduced LJ viscosity, where $\eta^* = \eta \sigma^3 / \epsilon / \tau$
- charge = reduced LJ charge, where $q^* = q / (4 \pi \epsilon_0 \sigma \epsilon)^{1/2}$
- dipole = reduced LJ dipole, moment where $\mu^* = \mu / (4 \pi \epsilon_0 \sigma^3 \epsilon)^{1/2}$
- electric field = force/charge, where $E^* = E (4 \pi \epsilon_0 \sigma \epsilon)^{1/2} \sigma / \epsilon$

For style *real*, these are the units:

- mass = grams/mole
- distance = Angstroms
- time = femtoseconds
- energy = Kcal/mole
- velocity = Angstroms/femtosecond
- force = Kcal/mole–Angstrom
- temperature = degrees K

- pressure = atmospheres
- viscosity = Poise
- charge = multiple of electron charge (+1.0 is a proton)
- dipole = charge*Angstroms
- electric field = volts/Angstrom

For style *metal*, these are the units:

- mass = grams/mole
- distance = Angstroms
- time = picoseconds
- energy = eV
- velocity = Angstroms/picosecond
- force = eV/Angstrom
- temperature = degrees K
- pressure = bars
- viscosity = Poise
- charge = multiple of electron charge (+1.0 is a proton)
- dipole = charge*Angstroms
- electric field = volts/Angstrom

For style *si*, these are the units:

- mass = kilograms
- distance = meters
- time = seconds
- energy = Joules
- velocity = meters/second
- force = Newtons
- temperature = degrees K
- pressure = Pascals
- viscosity = Pascal*second
- charge = Coulombs
- dipole = Coulombs*meters
- electric field = volts/meter

For style *cgs*, these are the units:

- mass = grams
- distance = centimeters
- time = seconds
- energy = ergs
- velocity = cm/second
- force = dynes
- temperature = degrees K
- pressure = dyne/cm² or barye = 1.0e−6 bars
- viscosity = Poise
- charge = statcoulombs or esu
- dipole = statcoul−cm = 10¹⁸ debye
- electric field = statvolt/cm or dyne/esu

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.

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 *equal* or *atom*

```

delete = no args
index args = one or more strings
loop args = N = integer size of loop
world args = one string for each partition of processors
universe args = one or more strings
uloop args = N = integer size of loop
equal or atom args = one formula containing numbers, thermo keywords, math operations, group
  numbers = 0.0, 100, -5.4, 2.8e-4, etc
  thermo keywords = vol, ke, press, etc from thermo\_style
  math operations = (), -x, x+y, x-y, x*y, x/y, x^y,
                    sqrt(x), exp(x), ln(x), log(x),
                    sin(x), cos(x), tan(x), asin(x), acos(x), atan(x),
                    ceil(x), floor(x), round(x)
  group functions = count(group), mass(group), charge(group),
                    xcm(group,dim), vcm(group,dim), fcm(group,dim),
                    bound(group,xmin), gyration(group)
  atom value = mass[N], x[N], y[N], z[N],
               vx[N], vy[N], vz[N], fx[N], fy[N], fz[N]
  atom vector = mass[], x[], y[], z[],
               vx[], vy[], vz[], fx[], fy[], fz[]
  compute references = c_ID, c_ID[2], c_ID[N], c_ID[N][2], c_ID[], c_ID[][2]
  fix references = f_ID, f_ID[2], f_ID[N], f_ID[N][2], f_ID[], f_ID[][2]
  variable references = v_abc, v_abc[N], v_abc[]

```

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 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
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 *equal* and *atom* ARE redefined each time the command is encountered. This only changes their associated formula if the formula contains a substitution for another variable, e.g. `$x`. But that can be useful, for example, in a loop.

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* option 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* option 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. 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.

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 operations, group functions, atom values, atom vectors, compute references, fix references, and references to other variables.

Number	0.2, 100, 1.0e20, -15.4, etc
Thermo keywords	vol, pe, ebond, etc
Math operations	(), -x, x+y, x-y, x*y, x/y, x^y, sqrt(x), exp(x), ln(x), log(x), sin(x), cos(x), tan(x), asin(x), acos(x), atan(x), ceil(x), floor(x), round(x)
Group functions	count(ID), mass(ID), charge(ID), xcm(ID,dim), vcm(ID,dim), fcm(ID,dim), bound(ID,dir), gyration(ID)
Atom values	mass[N], x[N], y[N], z[N], vx[N], vy[N], vz[N], fx[N], fy[N], fz[N]
Atom vectors	mass[], x[], y[], z[], vx[], vy[], vz[], fx[], fy[], fz[]
Compute references	c_ID, c_ID[2], c_ID[N], c_ID[N][2], c_ID[], c_ID[][2]
Fix references	f_ID, f_ID[2], f_ID[N], f_ID[N][2], f_ID[], f_ID[][2]
Other variables	v_abc, v_abc[N], v_abc[]

Note that formula elements that contain empty brackets, such as an atom vector, produce per-atom values. All other formula elements produce a global value.

A formula for equal-style variables cannot use any formula element that produces per-atom values. A formula for an atom-style variable can use formula elements that produce either global values or per-atom values.

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 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 operations are written in the usual way, where the "x" and "y" in the examples above can be another section of the formula. Operators are evaluated left to right and have the usual precedence: unary minus before exponentiation ("^"), exponentiation before multiplication and division, and multiplication and division before addition and subtraction. Parenthesis can be used to group one or more portions of a formula and enforce a desired order of operations. Additional math operations can be specified as keywords followed by a parenthesized argument, e.g. sqrt(v_ke). Note that ln() is the natural log; log() is the base 10 log. The ceil(), floor(), and round() operations 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.

Group functions take one or two arguments in a specific format. 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 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 [fix gyration](#) command for a definition of the formula.

Atom values take a single integer argument from 1-N, which is the desired atom-ID, e.g. x[243]., which means use the x coordinate of the atom with ID=243.

Atom vectors use empty brackets, i.e. they take no argument. They generate one value per atom, so that a

reference like `x[]` means the x-coord of each atom will be used when evaluating the variable.

Compute references access one or more quantities calculated by a [compute](#). The ID in the reference should be replaced by the actual ID of the compute defined elsewhere in the input script. See the doc pages for individual computes to see which ones calculate global versus per-atom quantities. If the compute reference contains empty brackets, then per-atom values calculated by the compute are accessed. Otherwise a single value (global or per-atom) calculated by the compute is accessed. If a variable containing a compute is evaluated 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".

The different kinds of compute references are as follows. M is a positive integer \leq the number of vector values calculated by the compute. N is a global atom ID (positive integer).

<code>c_ID</code>	scalar value of a global compute
<code>c_ID[2]</code>	vector component of a global compute
<code>c_ID[N]</code>	single atom's scalar value of a per-atom compute
<code>c_ID[N][M]</code>	single atom's vector component of a per-atom compute
<code>c_ID[]</code>	per-atom scalar from a per-atom compute
<code>c_ID[][M]</code>	per-atom vector component from a per-atom compute

Fix references access one or more quantities calculated by a [fix](#). The ID in the reference should be replaced by the actual ID of the fix defined elsewhere in the input script. See the doc pages for individual computes to see which ones calculate global versus per-atom quantities. If the fix reference contains empty brackets, then per-atom values calculated by the fix are accessed. Otherwise a single value (global or per-atom) calculated by the fix is accessed.

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. If a variable containing a fix is evaluated 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".

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_", and the word "compute" is replaced by "fix".

The current values of other variables can be accessed by prepending a "v_" to the variable name. This will cause that variable to be evaluated. Atom-style variables generate per-atom values; all other styles of variables generate a single scalar value.

The different kinds of variable references are as follows. N is a global atom ID (positive integer).

<code>v_ID</code>	scalar value of a non atom-style variable
<code>v_ID[N]</code>	single atom's scalar value from an atom-style variable
<code>v_ID[]</code>	per-atom value from an atom-style variable

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 will 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 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 it is an error.

However, your input script can also require variables to be evaluated before or after or inbetween runs. In this case, one of three things may happen.

(1) The variable may be evaluated accurately. If it contains references to a compute or fix, and these values were calculated and stored on the last timestep of a preceeding run, then they can be accessed and used by the variable.

(2) LAMMPS may not be able to evaluate the variable and generate an error. For example, if the variable formula requires a [compute](#) to be invoked, either directly or indirectly via a thermo keyword, then the variable cannot be evaluated before the first run has occurred. The general rule is that if a variable uses a value calculated by a compute, and the variable is not being evaluated during a run, then the variable will not invoke the compute. Instead, it can only use the value stored by the compute, which for accuracy requires that the compute was already invoked on the same timestep during a preceeding run.

Thus the way to get around this error 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.

Note that 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, this value may not be what you expect if the fix has not yet calculated the quantity of interest. 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, values of zero 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 to 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)
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 3.0 sum yes units box
velocity flow scale 300.0
velocity flow ramp lattice 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.

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, regardless 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 will always give the same set of velocities, independent of how many processors are used. However, the 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 on parallel machines that support parallel I/O for output.

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 set, etc).

Related commands:

[restart](#), [read_restart](#)

Default: none