

Table of Contents

<u>LAMMPS Documentation</u>	1
<u>1. Introduction</u>	2
<u>2. Getting Started</u>	9
<u>3. Commands</u>	16
<u>4. How-to discussions</u>	20
<u>5. Example problems</u>	25
<u>6. Performance & scalability</u>	27
<u>7. Additional tools</u>	28
<u>8. Modifying & extending LAMMPS</u>	31
<u>9. Errors</u>	40
<u>10. Future and history</u>	65
<u>angle coeff command</u>	67
<u>angle style command</u>	68
<u>atom style command</u>	69
<u>bond coeff command</u>	70
<u>bond style command</u>	72
<u>boundary command</u>	74
<u>cd command</u>	75
<u>clear command</u>	75
<u>create atoms command</u>	76
<u>create box command</u>	77
<u>delete atoms command</u>	77
<u>delete bonds command</u>	78
<u>dielectric command</u>	80
<u>dihedral coeff command</u>	80
<u>dihedral style command</u>	82
<u>dimension command</u>	83
<u>dipole command</u>	84
<u>displace atoms command</u>	85
<u>dump command</u>	86
<u>dump modify command</u>	88
<u>echo command</u>	89
<u>fix command</u>	90
<u>fix addforce command</u>	91
<u>fix aveforce command</u>	92
<u>fix com command</u>	93
<u>fix drag command</u>	93
<u>fix enforce2d command</u>	94
<u>fix freeze command</u>	94
<u>fix gran/diag command</u>	95
<u>fix gravity command</u>	96
<u>fix indent command</u>	97
<u>fix insert command</u>	98
<u>fix langevin command</u>	99
<u>fix lineforce command</u>	100
<u>fix modify command</u>	101
<u>fix msd command</u>	101
<u>fix npt command</u>	102

Table of Contents

<u>fix nve command</u>	104
<u>fix nve/gran command</u>	104
<u>fix nvt command</u>	105
<u>fix planeforce command</u>	106
<u>fix rigid</u>	106
<u>fix setforce command</u>	108
<u>fix shake style</u>	108
<u>fix spring style</u>	110
<u>fix temp/rescale command</u>	110
<u>fix tmd command</u>	111
<u>fix viscous command</u>	113
<u>fix volume/rescale command</u>	113
<u>fix wall/gran command</u>	114
<u>fix wall/93 command</u>	116
<u>fix wiggle command</u>	116
<u>group command</u>	117
<u>improper coeff command</u>	118
<u>improper style command</u>	120
<u>jump command</u>	121
<u>kspace modify command</u>	121
<u>kspace style command</u>	122
<u>lattice command</u>	124
<u>log command</u>	125
<u>mass command</u>	125
<u>neigh modify command</u>	126
<u>neighbor command</u>	128
<u>newton command</u>	129
<u>next command</u>	130
<u>orient command</u>	131
<u>origin command</u>	132
<u>pair coeff command</u>	132
<u>pair modify command</u>	136
<u>pair style command</u>	137
<u>processors command</u>	143
<u>read data command</u>	143
<u>read restart command</u>	149
<u>region command</u>	151
<u>replicate command</u>	152
<u>reset timestep command</u>	153
<u>restart command</u>	153
<u>run command</u>	154
<u>run style command</u>	155
<u>set type command</u>	157
<u>special bonds command</u>	158
<u>temp modify command</u>	160
<u>temper command</u>	160
<u>temperature command</u>	162
<u>thermo command</u>	163

Table of Contents

<u>thermo_modify command</u>	164
<u>thermo_style command</u>	165
<u>timestep command</u>	165
<u>undump command</u>	166
<u>unfix command</u>	167
<u>units command</u>	167
<u>variable command</u>	169
<u>velocity command</u>	170
<u>write_restart command</u>	172

LAMMPS Documentation

(17 Jan 2005 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 primary author of the code is [Steve Plimpton](mailto:sjplimp@sandia.gov), who can be contacted at sjplimp@sandia.gov. The [LAMMPS WWW Site](http://www.cs.sandia.gov/~sjplimp/lammps.html) at www.cs.sandia.gov/~sjplimp/lammps.html 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 us an [email](#) so we can improve the LAMMPS documentation.

[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 [Acknowledgements and citations](#)
2. [Getting started](#)
 - 2.1 [What's in the LAMMPS distribution](#)
 - 2.2 [Making LAMMPS](#)
 - 2.3 [Running LAMMPS](#)
 - 2.4 [Command-line options](#)
 - 2.5 [Screen output](#)
 - 2.6 [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 [Multiple simulations on different partitions](#)
 - 4.5 [Parallel tempering](#)
 - 4.6 [Granular models](#)
 - 4.7 [TIP3P water model](#)
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](#)

[Previous Section](#) – [LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#) – [Next Section](#)

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 [Acknowledgements 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, or granular systems using a variety of force fields and boundary conditions.

For examples of LAMMPS simulations, see the Publications page of the [LAMMPS WWW Site](#).

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

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

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

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

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

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

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

1.2 LAMMPS features

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

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

Force fields:

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

- pairwise vanderWaals and Coulombic potentials
- long-range Coulombics via Ewald summation or particle-particle particle-mesh (PPPM)
- CHARMM or AMBER force fields
- bond potentials (harmonic, FENE, nonlinear, bond-breaking)
- angle potentials (harmonic, cosine)
- dihedral potentials (harmonic, multi-harmonic)
- out-of-plane potentials (harmonic, cvff)
- embedded atom method (EAM) for metals and metal alloys
- frictional force fields for granular materials

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)

- constant NVE, NVT, NPT ensembles
- temperature control via rescaling, Nose/Hoover, or Langevin thermostatting
- pressure control via Nose/Hoover barostatting in 1 to 3 dimensions
- volume rescaling
- altered motion via velocity and force constraints
- harmonic (umbrella) constraint forces
- dragging of atoms to new positions
- SHAKE bond & angle constraints on small clusters of atoms
- rigid body motion of one or more groups of atoms
- wall constraints of various kinds
- targeted molecular dynamics (TMD) constraints
- gravity

Integrators:

(run, run_style, temper commands)

- velocity–Verlet integrator
 - rRESPA hierarchical time integrator
 - parallel tempering (replica exchange) across multiple simulations
 - multiple independent simulations simultaneously
-

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 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.

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, 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 [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:

- [Raster3d](#)
- [RasMol](#)
- [VMD](#)

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](#) 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.
- 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 Acknowledgements 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.doe.genomestolife.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 (www.cs.sandia.gov/~sjplimp/lammps.html). A paper describing the latest version of LAMMPS is in the works; when it appears in print, you can check the LAMMPS WWW Site for a more current citation.

If you send me information about your publication, I'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 primary author of LAMMPS is Steve Plimpton at Sandia National Labs. Others have made significant contributions to the code:

Ewald and PPPM solvers	Roy Pollock (LLNL)
rRESPA	Mark Stevens & Paul Crozier (Sandia)
NVT/NPT integrators	Mark Stevens (Sandia)
class 2 force fields	Eric Simon (Cray)
HTFN energy minimizer	Todd Plantenga (Sandia)
msi2lmp tool	Steve Lustig (Dupont),

	Mike Peachey & John Carpenter (Cray)
CHARMM force fields	Paul Crozier (Sandia)
2d Ewald/PPPM	Paul Crozier (Sandia)
granular force fields and BC	Leo Silbert & Gary Grest (Sandia)
multi-harmonic dihedral potential	Mathias Putz (Sandia)
EAM potentials	Stephen Foiles (Sandia)
parallel tempering	Mark Sears (Sandia)
Imp2cfg and Imp2traj tools	Ara Kooser, Jeff Greathouse, Andrey Kalinichev (Sandia)
FFT support for SGI SCLS (Altix)	Jim Shepherd (Ga Tech)
targeted molecular dynamics (TMD)	Paul Crozier (Sandia), Christian Burisch (Bochum Univeristy, Germany)
force tables for long-range Coulombics	Paul Crozier (Sandia)

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)
- Jim Belak (LLNL)

[Previous Section](#) – [LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#) – [Next Section](#)

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 [Running LAMMPS](#)

2.4 [Command-line options](#)

2.5 [Screen output](#)

2.6 [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 of the emails I get about build and run problems are not really about LAMMPS – 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](#). Note that doesn't include linking problems – that's a question for a local expert!

Also, 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 sjplimp@sandia.gov 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 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
```

If you get no errors and an executable like `Imp_linux` or `Imp_mac` is produced, you're done; it's your lucky day. The remainder of this section addressed the following topics: errors that occur when making LAMMPS, editing a new low-level Makefile.foo, how to make LAMMPS with and without packages, and additional build tips.

Errors that 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 -f Makefile.list linux
gmake -f Makefile.list mac
```

(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 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, I use 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. Makefile.foo needs to specify where the mpi.h file (-I switch) and the libmpi.a library (-L switch) is found.

On my Linux box, I use 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 link against 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 runs.

(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 my box was 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 me an email; I'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` command will read from gzipped files if you compile with `-DGZIP`. It requires that your Unix have certain include files available. 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.

(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.tflop, 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 `imp_foo` when the build is complete.

How to make LAMMPS with and without packages:

The source code for LAMMPS is structured as a large set of core files that are always used plus additional 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 packages by typing "make package". The current list of packages is as follows:

granular	force fields and boundary conditions for granular systems
kspace	long-range Ewald and particle-mesh (PPPM) solvers
metal	force fields for metallic systems
molecule	force fields for molecular systems

Any or all of these packages can be included or excluded when LAMMPS is built. The default is to include all packages. You may wish to exclude certain packages if you will never run certain kinds of simulations. This will produce a smaller executable which in some cases will also run a bit faster.

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-all" or "make no-all" to include/exclude all optional 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 not seen when LAMMPS is built. After you have included or excluded a package, you must re-make LAMMPS.

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-dir 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) Building for a Macintosh.

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

(4) Building for Windows.

I've never done this, but LAMMPS is just standard C++ with MPI and FFT calls. You should be able to use cygwin to build LAMMPS with a Unix-style make. Or you should be able to pull all the source files into Visual C++ (ugh) or some similar development environment and build it. Good luck – I can't help you on this one.

2.3 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.lammps` file with the same information. Note that this sequence of commands copied the LAMMPS executable (`lmp_linux`) to the directory with the input files. If you don't do this, LAMMPS may look for input files or create output files in the directory where the executable is, rather than where you run it from.

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.

2.4 Command-line options

At run time, LAMMPS recognizes several optional command-line switches which may be used in any order. E.g

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

These are the command-line options:

```
-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. 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 X value`

Specify a variable that will be defined for substitution purposes when the input script is read. X should be a single lower-case character from 'a' to 'z'. The value can be any string. Using this command-line option is equivalent to putting the line "variable X index value" at the beginning of the input script. See the [variable](#) command for more information.

2.5 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 reneighboring = 42
Dangerous reneighboring = 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 reneighboring 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.

2.6 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.

[Previous Section](#) – [LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#) – [Next Section](#)

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 for file names or user-specified 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 `$` characters. If the character following the `$` is "a" to "z", the two-characters sequence (e.g. `$x`) is replaced with the corresponding variable text. See the variable command for details.

(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) An argument with spaces can be enclosed in double quotes so it will be treated as a single argument. See the dump_modify command for an example.

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](#).

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](#), [orient](#), [origin](#), [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: [temperature](#), [temp_modify](#), [neighbor](#), [neigh_modify](#), [group](#), [timestep](#), [reset timestep](#), [run style](#).

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

Output options are set by these commands: [thermo](#), [dump](#), [restart](#).

(4) Run a simulation

3. Commands

A molecular dynamics simulation is run using the run 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 commands and their style options are part of specific LAMMPS packages. All packages are included in a LAMMPS build by default, but if you excluded a specific package when building LAMMPS, you cannot use the associated commands or styles. These dependencies are listed as Restrictions in the command's documentation.

Initialization:

atom style, boundary, dimension, newton, processors, units

Atom definition:

create atoms, create box, lattice, orient, origin, 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, special bonds

Settings:

dipole, group, mass, neigh modify, neighbor, reset timestep, run style, set type, temp modify, temperature, timestep, velocity

Fixes:

fix, fix modify, unfix

Output:

dump, dump modify, restart, thermo, thermo modify, thermo style, undump, write restart

Actions:

delete atoms, delete bonds, displace atoms, run, temper

Miscellaneous:

cd, clear, echo, jump, log, next, variable

3.5 Individual commands

This section lists all LAMMPS commands alphabetically. The [previous section](#) lists the same commands, grouped by category. Note that some commands and their style options are part of specific LAMMPS packages. All packages are included in a LAMMPS build by default, but if you excluded a specific package when building LAMMPS, you cannot use the associated commands or styles. These dependencies are listed as Restrictions in the command's documentation.

angle coeff	angle style	atom style	bond coeff	bond style	boundary
cd	clear	create atoms	create box	delete atoms	delete bonds
dielectric	dihedral coeff	dihedral style	dimension	dipole	displace atoms
dump	dump modify	echo	fix	fix modify	group
improper coeff	improper style	jump	kspace modify	kspace style	lattice
log	mass	neigh modify	neighbor	newton	next
orient	origin	pair coeff	pair modify	pair style	processors
read data	read restart	region	replicate	reset timestep	restart
run	run style	set type	special bonds	temp modify	temper
temperature	thermo	thermo modify	thermo style	timestep	undump
unfix	units	variable	velocity	write restart	

[Previous Section](#) – [LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#) – [Next Section](#)

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 [Multiple simulations on different partitions](#)
- 4.5 [Parallel tempering](#)
- 4.6 [Granular models](#)
- 4.7 [TIP3P water model](#)

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.

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.

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 Multiple simulations on different partitions

Use the `-procs` and `-in` command-line switches to launch LAMMPS on multiple partitions. See the variable, next, and jump commands for more details about the specific commands that are discussed below.

If you want to repeat a simulation with many different parameter settings, you could use the following input script. In this example, temperature is the parameter that is varied.

```
variable t index 0.8 0.85 0.9 0.95 1.0 1.05 1.1 1.15 1.2 1.25
...
read data.polymer
velocity all create $t 352839
fix 1 all nvt $t $t 100.0
dump 1 all atom 1000 dump.$t
...
run 100000
next t
jump in.run
```

The "variable" command defines a set of Lennard–Jones reduced temperatures. The \$t temperature variable is used in the "velocity", "fix", and "dump" commands to specify a particular temperature and store simulation output in a unique file. The "..." lines stand for other settings you wish to make, which could include \$t or other variables you define. As discussed in its documentation, the next command increments the \$t variable as each processor partition finishes its simulation. If the above script is stored in the file "in.run", then the jump command will restart the script with a new value for the \$t variable.

If you want to run a set of independent simulations, you could use this strategy. Assume you have 8 directories (run*), each with its own data.polymer and in.polymer input files.

Name this script "in.master" and store it in the directory that contains the run* sub-dirs:

```
variable d index run1 run2 run3 run4 run5 run6 run7 run8
cd $d
jump in.polymer
```

At the bottom of each in.polymer script put these lines:

```
cd ..
next d
clear
jump in.master
```

The clear command will re-initialize LAMMPS so that the partition of processors can run a new simulation.

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 proc 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

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

- atom_style granular
- fix nve/gran
- fix gravity
- thermo_style gran

Use one of these 3 pair potentials:

- 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 gran/diag
- fix insert
- 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 TIP3P model:

O charge = -0.834

H charge = 0.417

O mass = 15.9994

H mass = 1.008

LJ epsilon of O = 0.1521

LJ sigma of O = 3.15057

LJ epsilon of H = 0.046

LJ sigma of H = 0.400014

K of O-H bond = 450

r0 of O-H bond = 0.9572

K of H-O-H angle = 55

theta of H-O-H angle = 104.52

Note: If the LJ epsilon and sigma for H are set to 0.0, it corresponds to the original 1983 TIP3P model ([Jorgensen](#)).

(**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).

[Previous Section](#) – [LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#) – [Next Section](#)

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 [Tools section](#). MPEG versions of most of the xmovie animations are also viewable from the Examples page of the

LAMMPS WWW Site.

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

flow	Couette and Poiseuille flow in a 2d channel
indent	spherical indenter into a 2d solid
micelle	self-assembly of small lipid-like molecules into 2d bilayers
obstacle	flow around two voids in a 2d channel
pour	pouring of granular particles into a 3d box, then chute flow
crack	crack propagation in a 2d solid
friction	frictional contact of spherical asperities between 2d surfaces
melt	rapid melt of 3d LJ system
peptide	dynamics of a small solvated peptide chain (5-mer)
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
```

[Previous Section](#) – [LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#) – [Next Section](#)

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 benchmarks (on one processor) or in parallel. In parallel, each benchmark can be run as a fixed–size or scaled–size problem. For fixed–size benchmarking, the same 32K atom problem is run on various numbers of processors. For scaled–size benchmarking, the model size is increased with the number of processors. E.g. on 8 processors, a 256K–atom problem is run; on 1024 processors, a 32–million atom problem is run, etc.

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

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

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

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

[Previous Section](#) – [LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#) – [Next Section](#)

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.

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.

- [replicate](#)
- [restart2data](#)
- [data2xmovie](#)
- [chain](#)
- [micelle2d](#)
- [xmovie](#)
- [charmm2lmp](#)
- [msi2lmp](#)
- [amber2lammps](#)
- [lmp2arc](#)
- [lmp2cfg](#)
- [lmp2traj](#)

replicate tool

The file replicate.c takes a LAMMPS data file and replicates it into a larger system. The syntax for running the tool is

```
replicate options <infile > outfile
```

See the top of the replicate.c file for a discussion of the options. This tool is used by some of the [LAMMPS benchmarks](#) for creating larger systems to run scaled-size problems on multiple processors.

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
```

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.

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.

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](#).

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](#).

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!

charmm2lmp tool

The charmm2lmp sub-directory contains tools for converting files back-and-forth between the CHARMM MD code and LAMMPS.

The ch2lmp.pl Perl script runs CHARMM on a PDB file to assign force-field parameters and create LAMMPS input files. The mkpdf.f tool converts LAMMPS dump files back into PDB files that can be read-in to CHARMM for analysis. See the README file in the charmm2lmp sub-directory for more information.

This tool was written by Paul Crozier, Jessica Benjamin, and Mark Stevens at Sandia. Paul is the person to contact with questions (pscrozi@sandia.gov).

msi2lmp tool

The msi2lmp sub-directory contains a tool for creating LAMMPS input data files from Accelrys's Insight MD code (formerly MSI/Biosysm 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@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.

amber2lmp tool

The amber2lmp sub-directory contain 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.

Imp2arc tool

The Imp2arc sub-directory contains a tool for converting LAMMPS output files to the format for Accelrys's 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@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@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@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@sandia.gov).

[Previous Section](#) – [LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#) – [Next Section](#)

8. Modifying & extending LAMMPS

LAMMPS is designed in a modular fashion so as to be easy to modify and extend with new functionality. In this section, changes and additions users can make are listed along with some minimal instructions. Realistically, the best way to add a 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).

The new features described in this section require you to write a new C++ class (except for dump options, described below). This requires 2 files, one with source code (*.cpp) and a header file (*.h). Their contents are briefly discussed below. Enabling LAMMPS to invoke the new class is as simple as adding two definition lines to the style_user.h file, in the same syntax as the existing LAMMPS features are defined in the style.h file.

The power of C++ and its object-orientation is that usually, all the pieces of code and variables needed to define the new feature are contained in the 2 files you write, and thus shouldn't make the rest of the code 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 3.0 3.5
```

you simply need to 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 3.0 and 3.5 can be defined and processed by your new class.

Note that if you are using `Makefile.list` instead of `Makefile` to build LAMMPS, you will need to add the names of your new `.cpp` and `.h` file to `Makefile.list`.

Here is a list of the kinds of new features that can be added in this way:

- [Pairwise potentials](#)
- [Bond, angle, dihedral, improper potentials](#)
- [Dump options](#)
- [Thermodynamic output options](#)
- [Temperature computation options](#)
- [Region geometry options](#)
- [Fix options](#) which include integrators, temperature and pressure control, force constraints, boundary conditions, diagnostic output, etc
- [Atom options](#)

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 for each category will list the header file for the parent class that these styles are sub-classes of. Public variables in that file are ones used and set by the sub-classes which are also used by the parent class. Sometimes they are also used by the rest of LAMMPS. Virtual functions in the header file which are set = 0 are ones you must define in your new class to give it the functionality LAMMPS expects. Virtual functions that are not set to 0 are functions you can optionally define.

Here are some 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 me an email. We might be interested in adding it to the LAMMPS distribution.

Pairwise potentials

All classes that compute pairwise interactions are sub-classes of the Pair class. See the pair.h file for a list of methods this class defines.

Pair_lj_cut.cpp and pair_lj_cut.h are the simplest example of a Pair class. They implement the *lj/cut* style of the pair_style command.

Here is a brief description of the class methods in pair.h:

settings	reads the input script line with any arguments you define
set_coeff	reads pairwise coeffs from the input script or data file
init	insures all coeffs are set and performs setup computations
compute	the workhorse routine that computes the pairwise interactions
write & read_restart	writes/reads pair coeffs to restart files
compute_inner/middle/outer	versions of compute used

The inner/middle/outer routines are optional. Only a few of the pairwise potentials use these in conjunction with rRESPA as set by the run_style command.

Bond, angle, dihedral, improper potentials

All classes that compute molecular interactions are sub-classes of the Bond, Angle, Dihedral, and Improper classes. See the bond.h, angle.h, dihedral.h, and improper.h file for a list of methods these classes defines.

Bond_harmonic.cpp and bond_harmonic.h are the simplest example of a Bond class. Ditto for the harmonic forms of the angle, dihedral, and improper style commands. The bond_harmonic files implement the *harmonic* style of the bond_style command.

Here is a brief description of the class methods in bond.h, angle.h, etc:

set_coeff	reads coeffs from the input script or data file
init	insures all coeffs are set and performs setup computations
equilibrium_distance	length of bond, used by SHAKE
compute	the workhorse routine that computes the molecular interactions
write & read_restart	writes/reads coeffs to restart files

Dump options

Unlike the other styles described on this page, new dump features can be added without writing a new class. The dump command has a *custom* style that allows you to specify what information should be dumped with each atom. If the attribute you want to dump is not in the list, or if you define a new atom style with new attributes (e.g. atoms that store their own energy), here is how to dump it out in a snapshot file, via additions you make to the dump_custom.cpp and dump_custom.h file.

The dump_custom.cpp file has lines like the following one for FX. Add a new keyword to the list.

```
#define FX 12
```

In the `dump_custom` constructor, add 4 lines that define the attribute name (e.g. "fx"), `vnamei`, `vtypei`, and `pack_choicei` for your new option.

Add a new `pack` method to the `DumpCustom` class that stores your new atom quantity in the dump buffer, similar to all the other `pack_*` methods. The name of this new method is what you assigned to `pack_choicei`. You can do a modest amount of computation in this routine to write out precisely what you want – e.g. see the `pack_xs` routine, which scales the atom's x coordinate.

Add a prototype for your new method to the `dump_custom.h` file, like the other `pack_*` methods.

When you re-make LAMMPS, your new option should now be useable via the `dump custom` command.

Similar to the other styles in this section, you can also create new dump styles (like `atom`, `velocity`, `bond`) as new classes, if modifying the `dump custom` command is not sufficient for your needs. These are sub-classes of the `Dump` class. See the `dump.h` file for a list of methods these classes defines.

`Dump_velocity.cpp` and `dump_velocity.h` are the simplest example of a `Dump` class. They implement the *velocity* style of the `dump` command.

Here is a brief description of the class methods in `dump.h`:

<code>write_header</code>	writes the header section of each snapshot
<code>count</code>	counts the number of snapshot lines to be written out
<code>pack</code>	packs atom information into the dump buffer
<code>write_data</code>	writes out a buffer in the dump format

Thermodynamic output options

All classes that compute and print thermodynamic information to the screen and log file are sub-classes of the `Thermo` class. See the `thermo.h` file for a list of methods these classes defines.

`Thermo_one.cpp` and `thermo_one.h` are the simplest example of a `Thermo` class. They implement the *one* style of the `thermo` command.

Here is a brief description of the class methods in `thermo.h`:

header	writes the header to the thermodynamic output
compute	computes current thermodynamics of the system

Temperature computation options

All classes that compute the temperature of the system are sub-classes of the Temperature class. See the temperature.h file for a list of methods these classes defines. Temperatures are computed by LAMMPS when velocities are set, when thermodynamics are computed, and when temperature is controlled by various thermostats like the fix nvt or fix langevin commands.

Temp_full.cpp and temp_full.h are the simplest example of a Temperature class. They implement the *full* style of the temperature command.

Here is a brief description of the class methods in temperature.h:

init	setup the temperature computation
compute	compute and return temperature

Region geometry options

All classes that define geometric regions are sub-classes of the Region class. See the region.h file for a list of methods these classes defines. Regions are used elsewhere in LAMMPS to group atoms, delete atoms to create a void, insert atoms in a specified region, etc.

Region_sphere.cpp and region_sphere.h are the simplest example of a Region class. They implement the *sphere* style of the region command.

Here is a brief description of the single class method required:

match	determine whether a point is in the region
-------	--

Fix options

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 manipulation, and output, is a "fix". This includes time integration (update of velocity and coordinates), force constraints (SHAKE or walls), and diagnostics (compute a diffusion coefficient). See the `fix.h` file for a list of methods these classes defines.

There are dozens of fix options in LAMMPS; choose one as a template that is similar to what you want to implement. They can be as simple as zeroing out forces (see [`fix enforce2d`](#) which corresponds to the *enforce2d* style) or as complicated as applying SHAKE constraints on bonds and angles (see [`fix shake`](#) which corresponds to the *shake* style) which involves many extra computations.

Here is a brief description of the class methods in `fix.h`:

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

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`. If the fix needs to store information for each atom that persists from timestep to timestep, it can manage that memory and migrate it 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 a 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.

Atom options

All classes that define an atom style are sub-classes of the Atom class. See the atom.h file for a list of methods these classes defines. The atom style determines what quantities are associated with an atom in a LAMMPS simulation. If one of the existing atom styles does not define all the arrays you need to store with an atom, then a new atom class can be created.

Atom_atomic.cpp and atom_atomic.h are the simplest example of an Atom class. They implement the *atomic* style of the atom_style command.

Here is a brief description of the class methods in atom.h:

grow	allocate memory for each array associated with an atom
memory_usage	report how much memory is used
copy	copy info for one atom to another atom's array location
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
pack_restart	store all an atom's info to write to a restart file
unpack_restart	retrieve an atom's info from the buffer
size_restart	size of atom's info to write to a restart file
unpack_data	extract atom info from lines of data file
create_one	create an atom of this style
unpack_vels	extract atom velocity info from lines of data file

All of these methods are mandatory to define, except for unpack_vels.

Foo, Morefoo, and Maxfoo, J of Classic Potentials, 75, 345 (1997).

[Previous Section](#) – [LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#) – [Next Section](#)

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.

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 us an email.

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, we'd like to know about it via email.

First, check the "New features and bug fixes" section of the LAMMPS WWW site to see if the bug has already been reported or fixed.

If not, the most useful thing you can do for us 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.

Send an email that describes the problem and any ideas you have as to what is causing it or where in the code the problem might be. We'll request your input script and data files if necessary.

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.

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 coefficientss 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 coefficientss 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 coefficientss must be set in the data file or by the pair coeff command before running a simulation.

Angle atoms N N N missing on proc N at step N

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 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_type

The set_type 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 coeffs are not set

No angle coefficients have been assigned in the data file or via the angle_coeff command.

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.

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.

Atom style granular must perform 3d simulations
Atom style granular cannot be used with 2d simulations, because the pairwise potentials are inherently 3d.

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.

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 principal moments
Fix rigid did not compute the principal moments of inertia of a rigid group of atoms correctly.

Bad separation parameter
Kspace_modify value for the separation keyword must be ≥ 0.0 .

Bad slab/volume parameter
Kspace_modify value for the slab/volume keyword must be ≥ 2.0 .

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.

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_type
The set_type 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 N N missing on proc N at step N
One or more 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 coeffs are not set
No bond coefficients have been assigned in the data file or via the bond_coeff command.

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_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.

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.

Can only use fix gran/diag with atom style granular
Self-explanatory.

Can only use fix gran/diag with granular pair style
Self-explanatory.

Can only use fix nve/gran with atom style granular
Self-explanatory.

Can only use fix wall/gran with atom style granular
Self-explanatory.

Can only wiggle zcylinder wall in z dim
The Self-explanatory.

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 compute PPPM X grid spacing
LAMMPS failed to compute a valid PPPM grid spacing in the x dimension.

Cannot compute PPPM Y grid spacing
LAMMPS failed to compute a valid PPPM grid spacing in the y dimension.

Cannot compute PPPM Z grid spacing
LAMMPS failed to compute a valid PPPM grid spacing in the z dimension.

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 create_box until atom_style is defined
Self-explanatory.

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

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

Cannot fix npt on a non-periodic dimension
Pressure can only be controlled on a dimension that is periodic.

Cannot fix volume/rescale on a non-periodic boundary
Volume can only be rescaled on a dimension that is periodic.

Cannot mix setfl/funcfl EAM files
The pair_eam pairwise force field can only use either `setfl` or `funcfl` formatted potential files, not a mixture of both.

Cannot open EAM potential file X
The specified EAM potential file cannot be opened. Check that the path and name are correct.

Cannot open file X

- The specified file cannot be opened. Check that the path and name are correct.
- Cannot open fix com file X*
The output file for the fix com command cannot be opened. Check that the path and name are correct.
- Cannot open fix gran/diag file X*
The output file for the fix gran/diag command cannot be opened. Check that the path and name are correct.
- Cannot open fix msd file X*
The output file for the fix msd command cannot be opened. Check that the path and name are correct.
- Cannot open fix tmd file X*
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 restart file*
The output restart file cannot be opened. Check that the path and name are correct and that disk space is available.
- Cannot perform set_type of disallowed interactions*
Cannot perform set_type of bonds when bonds are not defined. Ditto for angles, etc.
- 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_data until atom_style is defined*
Self-explanatory.
- 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 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 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 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 setup EAM potentials from data file*
The specification of EAM force field coefficients cannot be done from the data file. It can only be done using the pair_coeff command in the input script.
- 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 PPPM separation of 0.0 with dipoles*

If using an atom style that defines dipoles with kspace style ppm, you must define a non-zero dipole separation with the kspace_modify command.

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_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 rigid with atom style granular

This fix is not yet enabled for this atom style.

Cannot use fix shake with non-molecular system

Your choice of atom style does not have bonds.

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 region INF when box does not exist

Regions that extend to the box boundaries can only be used after the create_box command has been used.

Command-line variable already exists

Cannot use the -var command-line option to define the same variable more than once.

Could not create 3d FFT plan

The FFT setup in ppm failed.

Could not create 3d remap plan

The FFT setup in ppm failed.

Could not find delete_atoms group ID

A group 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 dump_modify ID

A dump ID used in the dump_modify command does not exist.

Could not find dump group ID

A group ID used in the dump command does not exist.

Could not find fix group ID

A group ID used in the fix 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_modify ID

A fix ID used in the fix_modify command does not exist.

Could not find fix_modify temp ID

A temperature ID used in the fix_modify command does not exist.

Could not find temp_modify ID

A temperature ID used in the temp_modify command does not exist.

Could not find temperature group ID

A group ID used in the temperature command does not exist.

Could not find thermo_modify temp ID

A temperature ID used in the thermo_modify command does not exist.

Could not find undump ID

A dump ID used in the undump command does not exist.

Could not find unfix ID
A fix ID used in the unfix command does not exist.

Could not find velocity group ID
A group ID used in the velocity command does not exist.

Could not find velocity temperature ID
A temperature ID used in the velocity command does not exist.

Could not open dump file
The output file for the dump command cannot be opened. Check that the path and name are correct.

Could not open input script
The input script file named in a command-line argument could not be opened.

Could not 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.

Could not open logfile
The LAMMPS log file named in a command-line argument cannot be opened. Check that the path and name are correct.

Could not open logfile X
The LAMMPS log file specified in the input script cannot be opened. Check that the path and name are correct.

Could not open new input file X
The input script file named in a jump command could not be opened. Check that the path and name are correct.

Could not 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.

Could not 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.

Could not 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.

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 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.

Create_box region ID does not exist
A region ID used in the create_box command does not exist.

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 overlap not yet implemented
This option is not yet implemented in LAMMPS.

Delete_atoms region ID does not exist
A region ID used in the delete_atoms command does not exist.

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.

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 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.

Dihedral atoms N N N N missing on proc N at step N

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 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_type

The set_type 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 coeffs are not set

No dihedral coefficients have been assigned in the data file or via the dihedral_coeff command.

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.

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.

Energy must be computed on tempering swap steps

You need to use the thermo command to insure that thermodynamics (including energy) is computed on the timesteps that tempering swaps are attempted.

Failed to allocate N bytes for array X

Your LAMMPS simulation has run out of memory. You need to run a smaller simulation or on more processors.

Failed to reallocate N bytes for array X

Your LAMMPS simulation has run out of memory. You need to run a smaller simulation or on more processors.

Fix command before simulation box is defined

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

Fix insert region ID does not exist

A region ID used in the fix insert command does not exist.

Fix langevin period must be > 0.0

The time window for temperature relaxation must be > 0

Fix npt periods must be > 0.0

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

Fix nvt period must be > 0.0

The time window for temperature relaxation must be > 0

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 volume/rescale must use group all

Volume can only be rescaled for the entire ensemble of atoms.

Fix wall/gran can only be used with granular pair style

Self-explanatory.

Granular pair styles do not use pair_coeff settings

The pair_coeff command cannot be used with granular force fields.

Gravity must point in -z to use with insert fix

The fix insert command assumes the theta angle for gravity is 180.0.

Group command before simulation box is defined

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

Group ID does not exist

A group ID used in the group command does not exist.

Group region ID does not exist

A region ID used in the group command does not exist.

Illegal \$ variable

The character following a \$ in the input script is not between "a" and "z".

Illegal angle style

The choice of angle style is unknown.

Illegal angle type index for SHAKE

Self-explanatory.

Illegal angle_style command

Self-explanatory. Check the input script.

Illegal atom mass for SHAKE

Mass specified in [fix shake](#) command must be > 0.0.

Illegal atom style

The choice of atom style is unknown.

Illegal atom type in EAM setfl command

Atom type specified in pair_style eam command does not match number of types in setfl potential file.

Illegal atom type in neighbor exclusion list

Atom types specified in neigh_modify command must be from 1–N, where N is the number of atom types.

Illegal atom type index for SHAKE

Self-explanatory. Check the fix shake command in the input script.

Illegal atom_style command

Self-explanatory. Check the input script.

Illegal bond style

The choice of bond style is unknown.

Illegal bond type index for SHAKE

Self-explanatory. Check the fix shake command in the input script.

Illegal bond_style command

Self-explanatory. Check the input script.

Illegal boundary command

Self-explanatory. Check the input script.

Illegal cd command

Self-explanatory. Check the input script.

Illegal clear command

Self-explanatory. Check the input script.

Illegal coeffs for this angle style

Cannot set class 2 coeffs in data file for this angle style.

Illegal coeffs for this dihedral style

Cannot set class 2 coeffs in data file for this dihedral style.

Illegal coeffs for this improper style

Cannot set class 2 coeffs in data file for this improper style.

Illegal command-line argument

One or more command-line arguments is invalid. Check the syntax of the command you are using to launch LAMMPS.

Illegal create_atoms command

Self-explanatory. Check the input script.

Illegal create_box command

Self-explanatory. Check the input script.

Illegal data file section: Angle Coeffs

Atom style does not allow angles.

Illegal data file section: AngleAngle Coeffs

Atom style does not allow impropers.

Illegal data file section: AngleAngleTorsion Coeffs

Atom style does not allow dihedrals.

Illegal data file section: AngleTorsion Coeffs

Atom style does not allow dihedrals.

Illegal data file section: Angles

Atom style does not allow angles.

Illegal data file section: Bond Coeffs

Atom style does not allow bonds.

Illegal data file section: BondAngle Coeffs

Atom style does not allow angles.

Illegal data file section: BondBond Coeffs

Atom style does not allow angles.

Illegal data file section: BondBond13Torsion Coeffs

Atom style does not allow dihedrals.

Illegal data file section: Bonds

Atom style does not allow bonds.

Illegal data file section: Dihedral Coeffs

Atom style does not allow dihedrals.

Illegal data file section: Dihedrals

Atom style does not allow dihedrals.

Illegal data file section: EndBondTorsion Coeffs

Atom style does not allow dihedrals.

Illegal data file section: Improper Coeffs

Atom style does not allow impropers.

Illegal data file section: Impropers

Atom style does not allow impropers.

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

Illegal delete_atoms command
 Self-explanatory. Check the input script.

Illegal delete_atoms group command
 Self-explanatory. Check the input script.

Illegal delete_atoms region command
 Self-explanatory. Check the input script.

Illegal delete_bonds command
 Self-explanatory. Check the input script.

Illegal dielectric command
 Self-explanatory. Check the input script.

Illegal dihedral style
 The choice of dihedral style is unknown.

Illegal dihedral_style command
 Self-explanatory. Check the input script.

Illegal dimension command
 Self-explanatory. Check the input script.

Illegal dipole command
 Self-explanatory. Check the input script.

Illegal displace_atoms command
 Self-explanatory. Check the input script.

Illegal displace_atoms ramp command
 Self-explanatory. Check the input script.

Illegal dump command
 Self-explanatory. Check the input script.

Illegal dump atom command
 Self-explanatory. Check the input script.

Illegal dump velocity command
 Self-explanatory. Check the input script.

Illegal dump frequency
 Dumps frequency must be 1 or greater.

Illegal dump style
 The choice of dump style is unknown.

Illegal dump_modify command
 Self-explanatory. Check the input script.

Illegal echo command
 Self-explanatory. Check the input script.

Illegal fix addforce command
 Self-explanatory. Check the input script.

Illegal fix aveforce command
 Self-explanatory. Check the input script.

Illegal fix com command
 Self-explanatory. Check the input script.

Illegal fix command
 Self-explanatory. Check the input script.

Illegal fix drag command
 Self-explanatory. Check the input script.

Illegal fix enforce2d command
 Self-explanatory. Check the input script.

Illegal fix freeze command

Self-explanatory. Check the input script.
Illegal fix gran/diag command
 Self-explanatory. Check the input script.
Illegal fix gravity command
 Self-explanatory. Check the input script.
Illegal fix indent command
 Self-explanatory. Check the input script.
Illegal fix insert command
 Self-explanatory. Check the input script.
Illegal fix langevin command
 Self-explanatory. Check the input script.
Illegal fix lineforce command
 Self-explanatory. Check the input script.
Illegal fix msd command
 Self-explanatory. Check the input script.
Illegal fix npt command
 Self-explanatory. Check the input script.
Illegal fix npt command
 Self-explanatory. Check the input script.
Illegal fix nve command
 Self-explanatory. Check the input script.
Illegal fix nve/gran command
 Self-explanatory. Check the input script.
Illegal fix nvt command
 Self-explanatory. Check the input script.
Illegal fix planeforce command
 Self-explanatory. Check the input script.
Illegal fix rigid command
 Self-explanatory. Check the input script.
Illegal fix setforce command
 Self-explanatory. Check the input script.
Illegal fix shake command
 Self-explanatory. Check the input script.
Illegal fix spring command
 Self-explanatory. Check the input script.
Illegal fix style
 The choice of fix style is unknown.
Illegal fix temp/rescale command
 Self-explanatory. Check the input script.
Illegal fix tmd command
 Self-explanatory. Check the input script.
Illegal fix viscous command
 Self-explanatory. Check the input script.
Illegal fix volume/rescale command
 Self-explanatory. Check the input script.
Illegal fix wall/gran command
 Self-explanatory. Check the input script.
Illegal fix wall/lj93 command
 Self-explanatory. Check the input script.
Illegal fix wiggle command
 Self-explanatory. Check the input script.

Illegal fix_modify command

Self-explanatory. Check the input script.

Illegal group command

Self-explanatory. Check the input script.

Illegal group ID in neigh_modify command

A group ID used in the neigh_modify command does not exist.

Illegal improper style

The choice of improper style is unknown.

Illegal improper_style command

Self-explanatory. Check the input script.

Illegal jump command

Self-explanatory. Check the input script.

Illegal kspace style

The choice of kspace style is unknown.

Illegal kspace_modify command

Self-explanatory. Check the input script.

Illegal kspace_style ewald command

Self-explanatory. Check the input script.

Illegal kspace_style pppm command

Self-explanatory. Check the input script.

Illegal lattice command

Self-explanatory. Check the input script.

Illegal log command

Self-explanatory. Check the input script.

Illegal mass command

Self-explanatory. Check the input script.

Illegal min_style command

Self-explanatory. Check the input script.

Illegal minimize command

Self-explanatory. Check the input script.

Illegal minimize style

The choice of minimize style is unknown.

Illegal neigh_modify command

Self-explanatory. Check the input script.

Illegal neighbor command

Self-explanatory. Check the input script.

Illegal newton command

Self-explanatory. Check the input script.

Illegal next command

Self-explanatory. Check the input script.

Illegal 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.

Illegal orient command

Self-explanatory. Check the input script.

Illegal origin command

Self-explanatory. Check the input script.

Illegal pair style

The choice of pair style is unknown.

Illegal pair_modify command

Self-explanatory. Check the input script.

Illegal pair_style command
Self-explanatory. Check the input script.

Illegal processors command
Self-explanatory. Check the input script.

Illegal read_data command
Self-explanatory. Check the input script.

Illegal read_restart command
Self-explanatory. Check the input script.

Illegal region command
Self-explanatory. Check the input script.

Illegal region cylinder command
Self-explanatory. Check the input script.

Illegal region style
The choice of region style is unknown.

Illegal replicate command
Self-explanatory. Check the input script.

Illegal reset_timestep command
Self-explanatory. Check the input script.

Illegal restart command
Self-explanatory. Check the input script.

Illegal run command
Self-explanatory. Check the input script.

Illegal run style
The choice of run style is unknown.

Illegal run_style command
Self-explanatory. Check the input script.

Illegal run_style respa command
Self-explanatory. Check the input script.

Illegal set_type command
Self-explanatory. Check the input script.

Illegal special_bonds command
Self-explanatory. Check the input script.

Illegal temp_modify command
Self-explanatory. Check the input script.

Illegal temper command
Self-explanatory. Check the input script.

Illegal temper command
Self-explanatory. Check the input script.

Illegal temperature command
Self-explanatory. Check the input script.

Illegal temperature ramp command
Self-explanatory. Check the input script.

Illegal temperature style
The choice of temperature style is unknown.

Illegal temperature_modify command
Self-explanatory. Check the input script.

Illegal thermo command
Self-explanatory. Check the input script.

Illegal thermo style
The choice of thermo style is unknown.

Illegal thermo_modify command

Self-explanatory. Check the input script.

Illegal thermo_style command

Self-explanatory. Check the input script.

Illegal timestep command

Self-explanatory. Check the input script.

Illegal type for dipole set

Dipole command must set a type from 1–N where N is the number of atom types.

Illegal type for mass set

Mass command must set a type from 1–N where N is the number of atom types.

Illegal type in set_type command

Type used in set_type command must be from 1–N where N is the number of atom types (bond types, angle types, etc).

Illegal undump command

Self-explanatory. Check the input script.

Illegal unfix command

Self-explanatory. Check the input script.

Illegal units command

Self-explanatory. Check the input script.

Illegal variable command

Self-explanatory. Check the input script.

Illegal variable in command-line argument

Command-line arg –var must set a variable from "a" to "z".

Illegal variable in variable command

Variable command in input script must set a variable from "a" to "z".

Illegal vector in dump custom command

Atom vector specified in dump custom command is not recognized.

Illegal velocity command

Self-explanatory. Check the input script.

Illegal velocity command

Self-explanatory. Check the input script.

Illegal velocity ramp command

Self-explanatory. Check the input script.

Illegal write_restart command

Self-explanatory. Check the input script.

Improper atoms N N N N missing on proc N at step N

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 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_type

The set_type 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 coeffs are not set

No improper coefficients have been assigned in the data file or via the improper_coeff command.

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.

Impropers assigned incorrectly

Impropers read in from the data file were not assigned correctly to atoms. This means there is something invalid about the topology definitions.

Impropers defined but no improper types

The data file header lists improper but no improper types.

Incorrect args in `displace_atoms` options

Self-explanatory. Check the input script.

Incorrect args in `dump_modify` command

Self-explanatory. Check the input script.

Incorrect args in `fix insert` options

Self-explanatory. Check the input script.

Incorrect args in `fix_modify` command

Self-explanatory. Check the input script.

Incorrect args in `kpace_modify` command

Self-explanatory. Check the input script.

Incorrect args in `neigh_modify` command

Self-explanatory. Check the input script.

Incorrect args in `pair_coeff` command

Self-explanatory. Check the input script.

Incorrect args in `pair_modify` command

Self-explanatory. Check the input script.

Incorrect args in `pair_style` command

Self-explanatory. Check the input script.

Incorrect args in `region` options

Self-explanatory. Check the input script.

Incorrect args in `temp_modify` command

Self-explanatory. Check the input script.

Incorrect args in `temperature` options

Self-explanatory. Check the input script.

Incorrect args in `thermo_modify` command

Self-explanatory. Check the input script.

Incorrect args in `velocity` options

Self-explanatory. Check the input script.

Incorrect atom format in data file

Number of values per atom line in the data file is not consistent with the atom style.

Incorrect atom format in TMD target file

Number of values per line in the TMD target file is not 4.

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.

Insertion region extends outside simulation box

Region specified with `fix insert` command extends outside the global simulation box.

Insufficient Jacobi rotations for rigid body

Eigensolve for rigid body was not sufficiently accurate.

Invalid flag in header of restart file

Value read from beginning of restart file is not recognized.

KSpace style has not yet been set

Cannot use `kpace_modify` command until a `kpace` style is set.

KSpace style is incompatible with Pair style

Setting a `kpace` style requires that a pair style with a long-range Coulombic component be selected.

Lattice style incompatible with dimension

2d simulation can use `sq`, `sq2`, or `hex` lattice. 3d simulation can use `sc`, `bcc`, or `fcc` lattice.

Lost atoms via displacement: original N current N

Moving atoms via the `displace_atoms` command lost one or more atoms.

Lost atoms: original N current N

A thermodynamic computation has detected lost atoms.

Marsaglia RNG cannot use 0 seed

The random number generator use for the `fix langevin` command cannot use 0 as an initial seed.

Mass command before simulation box is defined

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

Min_HTFN not yet implemented

This feature has not yet been implemented in LAMMPS.

Minimize command before simulation box is defined

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

More than one freeze fix

You can only define one freeze fix.

More than one shake fix

You can only define one SHAKE fix.

Must define angle_style before Angle Coeffs

You must use an `angle_style` command before reading a data file that defines Angle Coeffs.

Must define angle_style before BondAngle Coeffs

You must use an `angle_style` command before reading a data file that defines Angle Coeffs.

Must define angle_style before BondBond Coeffs

You must use an `angle_style` command before reading a data file that defines Angle Coeffs.

Must define bond_style before Bond Coeffs

You must use a `bond_style` command before reading a data file that defines Bond Coeffs.

Must define dihedral_style before AngleAngleTorsion Coef

You must use a `dihedral_style` command before reading a data file that defines AngleAngleTorsion Coeffs.

Must define dihedral_style before AngleTorsion Coeffs

You must use a `dihedral_style` command before reading a data file that defines AngleTorsion Coeffs.

Must define dihedral_style before BondBond13Torsion Coef

You must use a `dihedral_style` command before reading a data file that defines BondBond13Torsion Coeffs.

Must define dihedral_style before Dihedral Coeffs

You must use a `dihedral_style` command before reading a data file that defines Dihedral Coeffs.

Must define dihedral_style before EndBondTorsion Coeffs

You must use a `dihedral_style` command before reading a data file that defines EndBondTorsion Coeffs.

Must define dihedral_style before MiddleBondTorsion Coef

You must use a `dihedral_style` command before reading a data file that defines MiddleBondTorsion Coeffs.

Must define improper_style before AngleAngle Coeffs

You must use an `improper_style` command before reading a data file that defines AngleAngle Coeffs.

Must define improper_style before Improper Coeffs

You must use an `improper_style` command before reading a data file that defines Improper Coeffs.

Must define pair_style before Pair Coeffs

- You must use an 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 insert*
 Self-explanatory.
- 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 insert*
 Self-explanatory.
- Must use a molecular atom style with fix rigid molecule*
 Self-explanatory.
- Must use a molecular atom style with neigh_modify exclude molecule*
 Self-explanatory.
- Must use a z-axis cylinder with fix insert*
 The axis of the cylinder region used with the fix insert command must be oriented along the z dimension.
- Must use atom style granular with lj units*
 Self-explanatory.
- Must use atom style granular with pair style granular*
 Self-explanatory.
- Must use atom style granular with thermo style gran*
 Self-explanatory.
- Must use atom types * * with EAM setfl files*
 The pair_style eam command with a setfl potential file must use 2 atom types of *,* instead of i,j.
- Must use charged atom style with this pair style*
 The atom style being used does not allow atoms to have assigned charges. Hence it will not work with this choice of pair style.
- Must use fix freeze with atom style granular*
 Self-explanatory.
- Must use fix gravity with atom style granular*
 Self-explanatory.
- Must use fix insert with atom style granular*
 Self-explanatory.
- Must use gravity fix with insert fix*
 Insertion of granular particles must be done under the influence of gravity.
- Must use region with side = in with fix insert*
 Self-explanatory.

Must use thermo style gran with atom style granular

Self-explanatory.

Needed topology not in data file

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

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.

Next command for multiple partitions not yet implemented

This option is not yet implemented in LAMMPS.

Next command has illegal variable

The variable specified by the next command is not valid or has not been defined with a variable command.

Next command has proc variable

The variable specified by the next command must be of index type.

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 atoms to compute diffusion for

The fix msd command has no atoms to compute on.

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 rigid bodies defined by fix rigid

Self-explanatory.

Non integer # of swaps in temper command

Swap frequency in temper command must evenly divide the total # of timesteps.

Non-orthogonal lattice vectors

Self-explanatory.

One or zero atoms in rigid body

Any rigid body defined by the fix rigid command must contain 2 or more atoms.

Orientation vectors are not right-handed

The 3 vectors defined by the orient command must form a right-handed coordinate system.

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 usually because an atom has moved to far in a single timestep.

PPPM order cannot be greater than N

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 class2 not yet implemented

This option is not yet implemented in LAMMPS.

Pair coeffs are not set

Coefficients have not been set for all atom types.

Pair cutoff < Respa interior cutoff

One or more pairwise cutoffs are too short to use with the specified rRESPA cutoffs.

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 style is incompatible with DihedralCharmm

When using a dihedral style charmm, a pair style with a CHARMM component must also be selected, so that 1–4 pairwise coefficients are specified.

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_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.

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".

Proc grid in z != 1 for 2d simulation

There cannot be more than 1 processor in the z dimension of a 2d simulation.

Proc variable count doesn't match # of partitions

A proc-style variable must specify a number of values equal to the number of processor partitions.

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.

Quotes in a single arg

A single word should not be quoted in the input script; only a set of words with intervening spaces should be quoted.

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 <= the second cutoff.

Respa inner/middle/outer used with illegal pair style

Only a few pair potentials support the use of respa inner, middle, outer options.

Respa levels must be >= 1

Self-explanatory.

Respa middle cutoffs are invalid

The first cutoff must be \leq the second cutoff.

Respa not allowed with atom style granular
 Respa cannot be used with the granular atom style.

Reuse of dump ID
 A dump ID cannot be used twice.

Reuse of region ID
 A region ID cannot be used twice.

Reuse of temperature ID
 A temperature ID cannot be used twice.

Rigid fix must come before NPT fix
 NPT 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_style command before simulation box is defined
 The run_style command cannot be used before a read_data, read_restart, or create_box command.

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 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 fix
 NPT fix must be defined in input script after SHAKE fix, else the SHAKE fix contribution to the pressure virial is incorrect.

Set_type command before simulation box is defined
 The set_type command cannot be used before a read_data, read_restart, or create_box command.

Set_type command with no atoms existing
 No atoms are yet defined so the set_type command cannot be used.

Shake atoms N N N N missing on proc N at step N
 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 N N N missing on proc N at step N
 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 N N missing on proc N at step N
 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.

Substitution for undefined variable
 The variable specified with a \$ symbol in an input script command has not been previously defined with a variable command.

Temper command before simulation box is defined
 The temper command cannot be used before a read_data, read_restart, or create_box command.

Tempering fix ID is not defined
 The fix ID specified by the temper command does not exist.

Tempering fix is not valid

The fix specified by the temper command is not one that controls temperature (nvt or langevin).

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.

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.

Too large an atom type in create_atoms command
The atoms to be created by the create_atoms command must have a valid type.

Too many exponent bits for the lookup table
Table size specified via pair_modify command does not work with your machine's floating point representation.

Too many mantissa bits for the 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 masses for 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.

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.

Unknown atom style in restart file
The atom style stored in the restart file is not recognized by LAMMPS.

Unknown command: X
The command is not known to LAMMPS. Check the input script.

Unknown identifier in data file: X
A section of the data file cannot be read by LAMMPS.

Unknown section in data file: X
The keyword for a section of the data file is not recognized by LAMMPS.

Use of displace_atoms with undefined lattice
Must use lattice command with displace_atoms 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 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 temperature ramp with undefined lattice
If scale = lattice (the default) for the temperature ramp 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_modify temp with invalid fix style

The fix_modify temp command was used but the fix does not support the temp option.

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.

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.

Warnings:

FENE bond too long: N N N X

A FENE bond has stretched dangerously far. Its interaction strength will be truncated to attempt to prevent the bond from blowing up.

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.

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 N current N

A thermodynamic computation has detected lost atoms.

Mismatch between velocity and temperature groups

The temperature computation used by the velocity command will not be on the same group of atoms that velocities are being set for. This is probably not what you want.

More than one dump custom with a centro attribute : dt

More than one dump custom command has been issued that requests a centro-symmetry attribute be dumped. This could be inefficient because each time each of the dumps is invoked a neighbor list for all atoms will be built. It may be more efficient to perform this computation once in a single dump.

More than one msd fix

This will be computationally inefficient.

More than one rigid fix

This will be computationally inefficient.

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.

One or more respa levels compute no forces

This is computationally inefficient.

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 atom_style to restart file value

The atom 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 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

The total charge on all atoms on the system is not 0.0, which is not valid for Ewald or PPPM.

Temperature for NPT pressure 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.

[Previous Section](#) – [LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#) – [Next Section](#)

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 many features from previous parallel MD codes I developed. These include earlier versions of LAMMPS itself, Warp and ParaDyn for metals, and GranFlow for granular materials.

These are features in those codes that haven't yet made it into the current LAMMPS:

- energy minimizers (conjugate gradient and Hessian-free truncated Newton)
- class 2 force fields
- Monte Carlo bond-swapping for polymers
- torsional shear boundary conditions and temperature calculation
- deletion of created atoms that overlap
- per-atom energy, stress computation

These are additional 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!

- threshold options for dumps
 - hybrid potentials for multi-material systems (e.g. polymers on metals)
 - bond breaking and creation potentials
 - point dipole force fields
 - 3-body force fields for materials like Si or silica
 - modified EAM (MEAM) potentials for metals
 - dissipative particle dynamics (DPD) potentials and integrators
 - Brownian dynamics
 - pressure and energy tail corrections for pairwise interactions
 - Parinello-Rahman non-rectilinear simulation box
-

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

[LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#)

angle_coeff command

Syntax:

```
angle_coeff N args
```

- N = angle type (see asterik 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 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 asterik 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 asterik with no numeric values means all types from 1 to N. A leading asterik means all types from 1 to n (inclusive). A trailing asterik means all types from n to N (inclusive). A middle asterik means all types from m to n (inclusive).

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 no wild-card asterik can be used. 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
```

See the [angle_style](#) command for the formulas that use these coefficients. The [units](#) of each coefficient are shown in parenthesis.

For style *harmonic*, specify 2 coefficients:

angle_coeff command

- K (energy)
- theta0 (degrees)

For style *charmm*, specify 4 coefficients:

- K (energy)
- theta0 (degrees)
- K_UB (energy/distance²)
- r_UB (distance)

For style *cosine*, specify 1 coefficient:

- K (energy)

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

[LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#)

angle_style command

Syntax:

```
angle_style style
```

- style = *none* or *harmonic* or *charmm* or *cosine*

Examples:

```
angle_style harmonic
angle_style charmm
```

Description:

Set the formula that LAMMPS will use to compute angle interactions for triplets of atoms. The list of atom triplets are specified in the data or restart file and are read in by a [read_data](#) or [read_restart](#) command. The coefficients for the formula for each angle type can also be specified in those files or via the [angle_coeff](#) command. In all the formulas to follow, *theta* is the angle defined by the triplet of atoms.

A style of *none* means angle forces are not computed, even if angles are defined.

The *harmonic* style uses the potential

$$E = K(\theta - \theta_0)^2$$

where θ_0 is the equilibrium value of the angle, and K is a prefactor. K and θ_0 are coefficients defined for each angle type.

The *charmm* 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.

The *cosine* style uses the potential

$$E = K[1 + \cos(\theta)]$$

where K is defined for each angle type.

Restrictions:

Angle styles can only be set for `atom_style`s that allow angles to be defined.

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.

Related commands:

[angle coeff](#)

Default:

`angle_style none`

[LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#)

atom_style command

Syntax:

`atom_style style`

- `style` = *atomic* or *bond* or *angle* or *full* or *eam* or *granular*

Examples:

`atom_style command`


```
atom_style bond
atom_style eam
```

Description:

Define what style of atoms to use in a simulation. This determines what attributes are associated with particles in the simulation. 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 a style general enough to encompass all attributes should be selected. 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.

These are the attributes of the available styles:

- *atomic* = no molecular topology, no charge – e.g. atomic LJ systems
- *bond* = bonds, no charge – e.g. bead–spring polymers
- *angle* = bonds & angles, no charge – bead–spring polymers with chain stiffness
- *full* = full topology (2,3,4–body terms) & charge – all–atom polymers, biomolecules, protein, DNA
- *eam* = metal or alloy system with EAM potentials, no molecular topology
- *granular* = granular material with rotational properties

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 *bond*, *angle*, and *full* styles are part of the "molecular" package. The *granular* style is part of the "granular" package. The *eam* style is part of the "metal" package. They are only enabled if LAMMPS was built with that package. See the [Making LAMMPS](#) section for more info.

Related commands: none

Default: none

[LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#)

bond_coeff command

Syntax:

```
bond_coeff N args
```

- N = bond type (see asterik 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 command

Description:

Specify the 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 asterik 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 asterik with no numeric values means all types from 1 to N. A leading asterik means all types from 1 to n (inclusive). A trailing asterik means all types from n to N (inclusive). A middle asterik means all types from m to n (inclusive).

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 no wild-card asterik can be used. 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
```

See the bond_style command for the formulas that use these coefficients. The units of each coefficient are shown in parenthesis.

For style *harmonic*, specify 2 coefficients:

- K (energy/distance²)
- r0 (distance)

For style *fene*, specify 4 coefficients:

- K (energy/distance²)
- R0 (distance)
- epsilon (energy)
- sigma (distance)

For style *fene/expand*, specify 5 coefficients:

- K (energy/distance²)
- R0 (distance)
- epsilon (energy)
- sigma (distance)
- delta (distance)

For style *nonlinear*, specify 3 coefficients:

- epsilon (energy)
- r0 (distance)
- lamda (distance)

For style *hybrid*, the first coefficient sets the bond style and the remaining coefficients are those appropriate to that style. For example, these commands:

```
bond_coeff 1 fene 30.0 1.5 1.0 1.0
```

`bond_coeff` command

```
bond_coeff 2 harmonic 80.0 1.2
```

would make bonds of bond type 1 be computed with a *fene* potential with coefficients 30.0, 1.5, 1.0, 1.0 for K, R0, epsilon, sigma. Likewise bonds of bond type 2 would be computed with a *harmonic* potential with coefficients 80.0, 1.2 for K, r0.

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

[LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#)

bond_style command

Syntax:

```
bond_style style
```

- style = *none* or *harmonic* or *fene* or *fene/expand* or *nonlinear* or *hybrid*

Examples:

```
bond_style harmonic
bond_style fene
```

Description:

Set the formula(s) that LAMMPS will use to compute bond interactions for pairs of atoms. The list of atom pairs are specified in the data or restart file and are read in by a [read_data](#) or [read_restart](#) command. The coefficients for the formula for each bond type can also be specified in those files or via the [bond_coeff](#) command. In all the formulas to follow, r is the distance between the 2 atoms in the bond.

A style of *none* means bond forces are not computed, even if bond are defined.

The *harmonic* style uses the potential

$$E = K(r - r_0)^2$$

where r_0 is the equilibrium bond distance. K and r_0 are coefficients defined for each bond type.

The *fene* style uses the potential

[bond_style](#) command

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

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. K , R_0 , ϵ , and σ are coefficients defined for each bond type.

The *fene/expand* style is similar to *fene* except that an extra shift factor of Δ (positive or negative) is added to r to effectively change the bead size of the bonded atoms. The corresponding potential is

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

The first term now extends to $R_0 + \Delta$ and the 2nd term is cutoff at $2^{1/6}$ sigma + Δ . K , R_0 , ϵ , σ , and Δ are coefficients defined for each bond type.

The *nonlinear* 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 . ϵ , r_0 , and λ are coefficients defined for each bond type.

The *hybrid* style allows 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 `bond_coeff` command is where the assignment of style to type is made.

Restrictions:

Bond styles can only be set for atom styles that allow bonds to be defined.

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.

Related commands:

`bond_coeff`, `delete_bonds`

Default:

`bond_style none`

(**Kremer**) Kremer, Grest, J Chem Phys, 92, 5057 (1990).

(**Rector**) Rector, Van Swol, Henderson, Molecular Physics, 82, 1009 (1994).

[LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#)

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 volume rescaling (see the [fix_npt](#) and [fix_volume/rescale](#) 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

[LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#)

cd command

Syntax:

```
cd dir
```

- dir = directory to change to

Examples:

```
cd subl  
cd ../new2  
cd ..
```

Description:

Change the working directory. All subsequent LAMMPS commands that access files for reading or writing will use the new directory.

Restrictions:

If the specified directory does not exist, LAMMPS will not detect the error.

Related commands: none

Default: none

[LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#)

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 ([cd](#) command), log file status ([log](#) command), echo status ([echo](#) command), and input script variables ([variable](#) command).

Restrictions: none

Related commands: none

Default: none

[LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#)

create_atoms command

Syntax:

```
create_atoms type args ...
```

- type = atom type (1–N) of atoms to create on a lattice
- args = zero or more region IDs

Examples:

```
create_atoms 1 reg1 reg2  
create_atoms 3
```

Description:

This command creates atoms on a lattice as an alternative to reading in their coordinates via a [read_data](#) or [read_restart](#) command. A simulation box must already exist, which is created with the [create_box](#) command.

Before using this command, a lattice must be defined using the [lattice](#) command. If no regions are specified, the [create_atoms](#) command fills the entire simulation box with atoms on the lattice. If regions are specified, then the geometric volume is filled that is inside the simulation box and is also consistent with each of the regions. Thus if 2 regions interior to the simulation box are specified, only the volume of their intersection will be filled.

The [create_atoms](#) command can be used multiple times with different lattice orientations to create grain boundaries. Used 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.

Created atoms are assigned the specified atom type and a velocity of 0.0.

Restrictions:

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

Related commands:

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

Default: none

[LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#)

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_atoms 2 mybox
```

Description:

This command creates a simulation box that encloses the specified region. Thus a [region](#) command must first be used to define a geometric domain. If the region is not of style *block*, LAMMPS encloses it with a rectangular simulation box.

The argument N is the number of atom types that will be used in the simulation.

Restrictions:

An [atom_style](#) and [region](#) must have been previously defined to use this command.

Related commands:

[create_atoms](#), [region](#)

Default: none

[LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#)

delete_atoms command

Syntax:

```
delete_atoms style args
```

- style = *group* or *region* or *overlap*

group args = group-ID

region args = one or more region IDs

[create_box](#) command

`overlap args = distance (distance units)`

Examples:

```
delete_atoms group edge
delete_atoms region reg1 reg2
delete_atoms overlap 0.3
```

Description:

Delete the specified atoms. For style *group*, it is all atoms belonging to the group. For style *region*, it is any atom that is in all of the listed regions. For style *overlap*, pairs of atoms within the specified distance are searched for, and one of the 2 atoms is deleted. See the [units](#) command for a discussion of distance units.

This command can be used to carve out voids from a block of material.

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: none

Related commands:

[create_atoms](#)

Default: none

[LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#)

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 to the args
- 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 *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 force fields (pair, bond, etc) be setup before using it, so that cutoff lengths are initialized and inter-processor communication can be performed to coordinate the deleting of bonds.

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

[LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#)

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
```

[LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#)

dihedral_coeff command

Syntax:

```
dihedral_coeff N args
```

- N = dihedral type (see asterik form below)

dielectric command

- 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 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 asterik 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 asterik with no numeric values means all types from 1 to N. A leading asterik means all types from 1 to n (inclusive). A trailing asterik means all types from n to N (inclusive). A middle asterik means all types from m to n (inclusive).

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 no wild-card asterik can be used. 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
```

See the [dihedral_style](#) command for the formulas that use these coefficients. The [units](#) of each coefficient are shown in parenthesis.

For style *harmonic*, specify 3 coefficients:

- K (energy)
- d (+1 or -1)
- n (1,2,3,4,6)

For style *charmm*, specify 4 coefficients:

- K (energy)
- n (1,2,3,4,6)
- d (0 or 180 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.

For style *multiharmonic*, specify 5 coefficients:

- A1 (energy)
- A2 (energy)
- A3 (energy)
- A4 (energy)

- A5 (energy)

Restrictions:

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

An 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

[LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#)

dihedral_style command

Syntax:

```
dihedral_style style
```

- style = *none* or *harmonic* or *charmm* or *multiharmonic*

Examples:

```
dihedral_style harmonic
dihedral_style multiharmonic
```

Description:

Set the formula that LAMMPS will use to compute dihedral interactions for quadruplets of atoms. The list of atom quadruplets are specified in the data or restart file and are read in by a [read_data](#) or [read_restart](#) command. The coefficients for the formula for each dihedral type can also be specified in those files or by the [dihedral_coeff](#) command. In all the formulas to follow, *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.

A style of *none* means dihedral forces are not computed, even if dhedrals are defined.

harmonic The *harmonic* style uses the potential

$$E = K[1 + d \cos(n\phi)]$$

K, d, and n are coefficients defined for each dihedral type.

The *charmm* style uses the potential

$$E = K[1 + d \cos(n\phi)]$$

K, d, and n are coefficients defined for each dihedral type. Additionally, a weighting factor if defined (see the [dihedral coeff](#) command) which is applied to the pairwise LJ and Coulombic interaction between the 1st and 4th atom in the dihedral quadruplet.

The *multiharmonic* style uses the potential

$$E = \sum_{n=1,5} A_n \cos^{n-1}(\phi)$$

A1, A2, A3, A4, and A5 are coefficients defined for each dihedral type.

Restrictions:

Dihedral styles can only be set for atom styles that allow dihedrals to be defined.

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.

Related commands:

[dihedral coeff](#)

Default:

dihedral_style none

[LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#)

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.

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
```

[LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#)

dipole command

Syntax:

```
dipole I value
```

- I = atom type (see asterik form below)
- value = dipole

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.

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 asterik 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 asterik with no numeric values means all

types from 1 to N. A leading asterik means all types from 1 to n (inclusive). A trailing asterik means all types from n to N (inclusive). A middle asterik means all types from m to n (inclusive).

A line in a data file that specifies a dipole moment uses the exact same format as the arguments of the dipole command in an input script, except that no wild-card asterik 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).

Related commands: none

Default: none

[LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#)

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*

```
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)
```

- zero or more keyword/value pairs may be appended to the args

```
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 useful to move atoms a large distance before beginning a simulation. 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.

Distance units for the 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 to use lattice spacings as defined by the lattice command. The default is to use lattice units.

Care should be taken not to move atoms on top of other atoms. After the move, atoms are remapped to the periodic simulation box. In parallel, atoms should not be moved so far that they cross more than one processor's sub-domain, else they may be lost. If this is a problem, successive `displace_atom` commands can be used to move a greater distance.

Restrictions: none

Related commands: none

Default:

The option defaults are `units = lattice`.

[LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#)

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 *velocity* or *bond* 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
velocity args = none
bond args = none
custom args = list of atom attributes
    attributes = tag, type, x, y, z, xs, ys, zs, ix, iy, iz, vx, vy, vz,
                fx, fy, fz, q, mux, muy, muz, tqx, tqy, tqz, centro
    tag = atom ID
    type = atom type
    x,y,z = unscaled atom coordinates
    xs,ys,zs = scaled atom coordinates
    ix,iy,iz = box image that the atom is in in
    vx,vy,vz = atom velocities
    fx,fy,fz = forces on atoms
    q = atom charge
```

```

mux,muy,muz = orientation of dipolar atom
tqx,tqy,tqz = torque on dipolar atoms
centro = per-atom centro-symmetry parameter

```

Examples:

```

dump myDump all atom 100 dump.atom
dump 2 subgroup atom 50 dump.run
dump 3 all velocity 1000 dump.vels
dump 4ab all custom 100 dump.myforce tag type x y vx fx

```

Description:

Dump a snapshot of atom quantities to a file every so many timesteps. When a dump is defined, the file is opened. The file is closed when an undump command is used or when LAMMPS exits. Only information for atoms in the specified group is dumped. Because snapshot data is collected from multiple processors, the order of lines (typically one per atom) written into the dump file for a single snapshot is indeterminate.

Dumps are performed on timesteps that are a multiple of N , including timestep 0. If one run ends and another begins on a timestep that is a multiple of N , only one snapshot is written.

The style determines what quantities are written to the file. Settings made via the dump modify command can alter the output format.

For style *atom*, atom coordinates are written to the file. For header settings *item* and *self* of the dump modify command, the atom ID and type are also written. For header setting *xyz*, only the atom type is included. By default, coordinates are in normalized units from 0.0 to 1.0. The *scale* setting of the dump modify command enables unnormalized coordinates to be written out. Because periodic boundary conditions are enforced only on timesteps when neighbor lists are rebuilt, the coordinates of some atoms may be slightly outside the simulation box.

For style *velocity*, atom velocities are written to the file along with the atom ID and type.

For style *bond*, the bond topology between atoms is written, in the same format they are specified in the data file read 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) are not written. Bonds that have been turned off (see the fix shake or delete bonds commands) 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 described above and will appear in the order specified. Be careful not to specify a quantity that is not defined for a particular simulation – e.g. *q* for atom style bond, since that atom style doesn't assign charges.

The *mux*, *muy*, *muz*, *tqy*, *txx*, *tqy* attributes are specific to dipolar systems defined with an atom style of *dipole*.

The *centro* attribute causes the centro-symmetry parameter to be computed for each atom in the dump group (Kelchner). For solid state systems this 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. Computing this quantity requires a neighbor list be constructed containing all the neighbors of each atom. This is performed each time the dump is performed. Thus it could be inefficient to dump this quantity too frequently or to have multiple dump commands, each with a *centro* attribute.

Dumps occur at the very end of a timestep, so atom attributes will include any effects due to fixes that are applied during the timestep.

See [this section](#) for information on how to modify LAMMPS to dump other kinds of per-atom quantities.

Restrictions:

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.

Related commands:

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

Default: none

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

[LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#)

dump_modify command

Syntax:

```
dump_modify dump-ID keyword value ...
```

- dump-ID = ID of dump to modify
- one or more keyword/value pairs may be appended
- keyword = *format* or *scale* or *image* or *header*

```
format value = C-style format to use when atom quantities are written
scale value = yes or no, default = yes
image value = yes or no, default = no
header value = item or self or xyz, default = item
```

Examples:

```
dump_modify 1 format "%d %d %20.15g %g %g" scale yes
dump_modify myDump image yes scale no
dump_modify 1 header xyz
```

Description:

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

Each dump style has 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.

The *scale* and *image* keywords apply only to dump atom commands. A scale value of *yes* means atom coords are written in normalized (scaled) 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 -3 with a normalized coord of 0.5 means the atom is in the center of the box, but has passed thru the box boundary 3 times and is really 3 box lengths to the left of its current coordinate.

The *header* keyword determines the file format for the snapshots. A value of *item* means each keyword is prefaced by "ITEM:" which is compatible with previous versions of LAMMPS. A value of *self* is a self-documenting format where each keyword is followed by the number of rows and columns of data that follow it. This makes it easy to write post-processing codes that parse the dump output. A value of *xyz* writes the dump file in the XYZ format used by other molecular modeling codes. For dump atom commands, each line will have 4 quantities: the atom type and unscaled coordinates. For dump custom commands, each line will still list the quantities it specifies.

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, header = item .

[LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#)

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.

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.

Each fix style has its own documentation page which describes its arguments and what it does. For example, see the [fix setforce](#) page for information on style *setforce*.

Fixes perform their operations at different stages of the timestep. If 2 or more fixes both operate at the same stage of the timestep, they are invoked in the order they were specified in the input script.

Specifying a new fix with the same ID as an existing fix effectively replaces the old fix (and its parameters) with the new fix. This can only be done if the new fix has the same style as the existing fix.

Fixes can be deleted with the [unfix](#) command. Note that this 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. 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!

Here is an alphabetic list of fix styles defined in LAMMPS:

- [fix addforce](#) – add a force to each atom
- [fix aveforce](#) – add an averaged force to each atom
- [fix com](#) – compute a center-of-mass
- [fix drag](#) – drag atoms towards a defined coordinate

- [fix enforce2d](#) – zero out z–dimension velocity and force
- [fix freeze](#) – freeze atoms in a granular simulation
- [fix gran/diag](#) – compute granular diagnostics
- [fix gravity](#) – add gravity to atoms in a granular simulation
- [fix indent](#) – impose force due to an indenter
- [fix insert](#) – add new atoms to a granular simulation
- [fix langevin](#) – Langevin temperature control
- [fix lineforce](#) – constrain atoms to move in a line
- [fix msd](#) – compute mean–squared displacement (i.e. diffusion coefficient)
- [fix npt](#) – constant NPT time integration via Nose/Hoover
- [fix nve](#) – constant NVE time integration
- [fix nve/gran](#) – NVE time integration for granular systems
- [fix nvt](#) – constant NVT time integration via Nose/Hoover
- [fix plane](#) – constrain atoms to move in a plane
- [fix rigid](#) – constrain one or more clusters of atoms to move as a rigid body
- [fix setforce](#) – set the force on each atom
- [fix shake](#) – SHAKE constraints on bonds and/or angles
- [fix spring](#) – apply harmonic spring force to atoms
- [fix temp/rescale](#) – temperature control by velocity rescaling
- [fix tmd](#) – guide a group of atoms to a new configuration
- [fix viscous](#) – viscous damping for granular simulations
- [fix volume/rescale](#) – density control by volume rescaling
- [fix wall/gran](#) – frictional walls for granular simulations
- [fix wall/lj93](#) – Lennard–Jones 9/3 walls
- [fix wiggle](#) – oscillate walls and frozen atoms

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.

The *freeze*, *gran/diag*, *gravity*, *insert*, *nve/gran*, and *wall/gran* styles are part of the "granular" package.

Related commands:

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

Default: none

[LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#)

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.

Restrictions: none

Related commands:

[fix setforce](#), [fix aveforce](#)

Default: none

[LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#)

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.

Restrictions: none

Related commands:

[fix aveforce command](#)

[fix setforce](#), [fix addforce](#)

Default: none

[LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#)

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.

Restrictions: none

Related commands: none

Default: none

[LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#)

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:

fix com command

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.

Restrictions: none

Related commands:

[fix spring](#)

Default: none

[LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#)

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.

Restrictions: none

Related commands: none

Default: none

[LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#)

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.

Restrictions:

Can only be used with atom_style granular.

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

[LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#)

fix gran/diag command

Syntax:

```
fix ID group-ID gran/diag nevery file zlayer
```

- ID, group-ID are documented in [fix](#) command
- gran/diag = style name of this fix command
- nevery = compute diagnostics every this many timesteps
- file = filename to store diagnostic info in
- zlayer = bin size in z dimension

Examples:

```
fix 1 all gran/diag 1000 tmp 0.9
```

Description:

Compute aggregate density, velocity, and stress diagnostics for a group of granular atoms as a function of z depth in the granular system. The results are written to 3 files named file.den, file.vel, and file.str. The z bins begin at the bottom of the system and extend upward with a thickness of zlayer for each bin. The quantities written to the file are averaged over all atoms in the bin.

Restrictions:

Can only be used with atom_style granular.

Related commands:

[atom_style granular](#)

Default: none

[LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#)

fix gravity command

Syntax:

```
fix ID group gravity style args
```

- ID, group are documented in [fix](#) command
- gravity = style name of this fix command
- style = *chute* or *spherical* or *gradient*

```
chute args = angle
    angle = angle in +x away from -z axis (in degrees)
spherical args = phi theta
    phi = azimuthal angle from +x axis (in degrees)
    theta = angle from +z axis (in degrees)
gradient args = phi theta phi_grad theta_grad
    phi = azimuthal angle from +x axis (in degrees)
    theta = angle from +z axis (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)
```

Examples:

```
fix 1 all gravity chute 24.0
fix 1 all gravity spherical 0.0 -180.0
fix 1 all gravity gradient 0.0 -180.0 0.0 0.1
```

Description:

Impose an additional force on each granular particle in the group due to gravity. The direction in which gravity operates is specified.

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. Style *spherical* allows an arbitrary 3d direction to be specified for the gravity vector. Style *gradient* allows the direction of the gravity vector to be 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 gravity vector would rotate thru 360 degrees every 10,000 timesteps. For the time-dependent case, the initial direction of the gravity vector is phi,theta at the time the fix is specified.

The strength of the acceleration due to gravity is 1.0 in LJ units, which are the only allowed units for granular systems.

Restrictions:

Can only be used with atom_style granular.

Related commands:

atom_style granular

Default: none

[LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#)

fix indent command

Syntax:

```
fix ID group-ID indent k R x y z vx vy vz
```

- ID, group-ID are documented in [fix](#) command
- indent = style name of this fix command
- k = force constant for indenter surface (force/distance units)
- R = radius of indenter (distance units)
- x,y,z = initial position of center of indenter
- vx,vy,vz = velocity of center of indenter (velocity units)

Examples:

```
fix 1 all indent 10.0 3.0 0.0 0.0 15.0 0.0 0.0 -1.0
```

Description:

Move a spherical indenter within a simulation box. The indenter repels all atoms if touches, so it can be used to push into a block of material or as an obstacle in a flow. The magnitude of the force exerted by the indenter on each atom is given by

$$F(r) = -k (r - R)$$

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 $r > R$. The center of the indenter moves during the simulation, based on its initial (x,y,z) position and the specified (vx,vy,vz).

Restrictions: none

Related commands: none

Default: none

[LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#)

fix insert command

Syntax:

```
fix ID group-ID insert N type seed keyword values ...
```

- ID, group-ID are documented in [fix](#) command
- insert = style name of this fix command
- N = # of atoms to insert
- type = atom type to assign to inserted atoms
- seed = random # seed
- one or more keyword/value pairs may be appended to args
- keyword = *region* or *diam* or *dens* or *vol* or *zrate* 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
zrate value = rate
    rate = z velocity at which insertion volume moves (velocity units)
vel values = vxlo vxhi vylo vyhi vz
    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 assigned to inserted particles (velocity units)
```

Examples:

```
fix 3 all insert 1000 2 29494 region myblock
fix 2 all insert 10000 1 19985583 region disk vol 0.33 100 zrate 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.

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 insert 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*.

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* value for option *vel* assigns a z-velocity 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. LAMMPS will make up to *Nattempt* tries to insert a new particle without overlaps. If it fails it prints a warning.

The *zrate* option allows the insertion volume to move in the z direction. This enables pouring particles from a successively higher height over time.

Restrictions:

Can only be used with *atom_style granular*. A gravity fix in the -z direction must be defined for use in conjunction with this fix.

Related commands:

[fix_gravity_region](#)

Default:

The option defaults are *diam* = 1.0 1.0, *dens* = 1.0 1.0, *vol* = 0.25 50, *zrate* = 0.0, *vel* = 0.0 0.0 0.0 0.0 0.0.

[LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#)

fix langevin command

Syntax:

```
fix ID group-ID langevin Tstart Tstop damp seed xflag yflag zflag
```

- 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 = Langevin damping parameter (time units)
- seed = random # seed to use for white noise
- xflag,yflag,zflag = 0/1 for whether to apply to each dimension (optional)

Examples:

```
fix 3 boundary langevin 1.0 1.0 1000.0 699483
fix 1 all langevin 1.0 1.1 100.0 48279 0 1 1
```

Description:

Apply a Langevin thermostat to a group of atoms. Uniform random numbers are used to generate a white-noise term that is added to the force of each atom to keep them at a specified temperature. The desired temperature at each timestep is a ramped value during the run from *Tstart* to *Tstop*. 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 random *# seed* should be a non-zero integer with 1 to 8 digits. 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. Also, the state of the random number generator is not saved in a restart file. This means you cannot do exact restarts when a `fix langevin` command is used.

The last 3 arguments are flags that specify which dimensions to add langevin white noise to. A flag of 0 means do not add noise to that dimension. A flag of 1 means add noise. The default is 1 for all 3 dimensions. These flags are optional; use all 3 or none of them.

The way that temperature is computed by this fix can be changed by using the [fix_modify](#) command.

A langevin fix does not update the coordinates or velocities of its atoms. It is normally used with a fix of style *nve* that does that. A langevin fix should not normally be used on atoms that also have their temperature controlled by another fix – e.g. a [nvt](#) or [temp/rescale](#) fix.

Restrictions: none

Related commands:

[fix nvt](#), [fix temp/rescale](#), [fix_modify](#)

Default: none

[LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#)

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 it's motion will be along the linear direction specified by the vector (x,y,z). This is done by subtracting out components of force 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.

Restrictions: none

Related commands:

[fix planeforce](#)

Default: none

[LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#)

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*

temp value = temperature ID

Examples:

```
fix_modify 3 temp myTemp
```

Description:

Modify a parameter of a previously defined fix. Parameters are only relevant to particular fix styles.

The *temp* keyword is used to determine how a fix computes temperature. The specified temperature ID must have been previously defined by the user via the [temperature](#) command. The default setting for *temp* is temperature ID = *default*. Fix styles that use the *temp* setting are [temp/rescale](#), [nvt](#), and [npt](#).

Restrictions: none

Related commands:

[fix, temperature](#)

Default:

The option defaults are temp = default.

[LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#)

fix msd command

Syntax:

```
fix ID group-ID msd N file
```

[fix_modify command](#)

- 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

Examples:

```
fix 1 all msd 100 diff.out
```

Description:

Compute the mean-squared displacement of the group of atoms every N steps, including all effects due to atoms passing thru periodic boundaries. The slope of the mean-squared displacement versus time is proportional to the diffusion coefficient of the diffusing atoms. The "origin" of the displacements is the position of each atom at the time the fix command was issued. Write the results to the specified file.

Restrictions: none

Related commands: none

Default: none

[LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#)

fix npt command

Syntax:

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

- 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 = frequency constant for temperature damping
- 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 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)
```

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 aniso 0.0 0.0 0.0 0.0 NULL NULL 1000.0
```

Description:

Perform constant NPT integration each timestep 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 desired temperature at each timestep is a ramped value during the run from T_{start} to T_{stop} . 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).

Regardless of what group is specified for this fix, a global pressure is computed for all atoms. Similarly, when the size of the simulation box is changed and atoms are scaled to new positions, all atoms are re-scaled.

By default, the temperature computed by this fix is also computed for all atoms, regardless of what group is specified. This is because the pressure contains a kinetic energy term which is derived from temperature, and the kinetic energy should be consistent with the virial term computed for all atoms. The way that temperature is computed can be changed by using the fix_modify command. LAMMPS will warn you if you choose to compute temperature on a subset of atoms.

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.

The pressure can be controlled in one of several styles, as specified by the *p-style* argument. Style *xyz* means couple all 3 dimensions together when pressure is computed (isotropic pressure), and dilate/contract the 3 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.

For style *aniso*, all 3 dimensions dilate/contract independently using their individual pressure components as the 3 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 all pressure styles, the simulation box stays rectangular in shape. Parinello–Rahman boundary conditions (tilted box) are not implemented in LAMMPS.

For all styles, the $Pdamp$ parameter operates like the $Tdamp$ parameter, determining the time scale on which pressure is relaxed.

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 nvt](#), [fix modify](#)

Default: none

[LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#)

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 updates on a group of atoms each timestep. V is volume; E is energy. This creates a system trajectory consistent with the microcanonical ensemble.

Restrictions: none

Related commands:

[fix nvt](#), [fix npt](#)

Default: none

[LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#)

fix nve/gran command

Syntax:

```
fix ID group-ID nve/gran
```

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

Examples:

```
fix 1 all nve/gran
```

Description:

Perform constant NVE updates each timestep on a group of atoms of atom style granular. V is volume; E is energy. Granular atoms store rotational information as well as position and velocity, so this integrator updates

translational and rotational degrees of freedom due to forces and torques.

Restrictions: none

Can only be used with atom_style granular.

Related commands:

atom_style granular

Default: none

[LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#)

fix nvt command

Syntax:

```
fix ID group-ID nvt Tstart Tstop Tdamp
```

- 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)

Examples:

```
fix 1 all nvt 300.0 300.0 100.0
```

Description:

Perform constant NVT integration each timestep using a Nose/Hoover temperature thermostat. V is volume; T is temperature. This creates a system trajectory consistent with the canonical ensemble.

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 way that temperature is computed by this fix can be changed by using the fix_modify command.

Restrictions: none

Related commands:

fix nve, fix npt, fix temp/rescale, fix langevin, fix_modify

Default: none

[LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#)

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 it's motion will be in the plane specified by the normal vector (x,y,z). This is done by subtracting out components 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.

Restrictions: none

Related commands:

[fix lineforce](#)

Default: none

[LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#)

fix rigid

Syntax:

```
fix ID group-ID rigid keyword values
```

- ID, group-ID are documented in [fix](#) command
- freeze = 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 a rigid body. This means that each timestep the total force and torque on the rigid body is computed and the coordinates and velocities of the atoms 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, temp/rescale, langevin).

For *single* the entire group of atoms is treated as one rigid body.

For *molecule*, each set of atoms in the group with a different molecule ID is treated as a rigid body.

For *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 ideally define one rigid fix which includes all the desired rigid bodies. LAMMPS will allow multiple rigid fixes to be defined, but it is more expensive.

The degrees-of-freedom removed by rigid bodies are accounted for in temperature and pressure computations. Similarly, the rigid body contribution to the pressure virial is also accounted for.

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.

Because this fix uses constant-energy integration, it means you cannot easily control the temperature of an ensemble of rigid bodies. You can try to use other fixes (langevin, temp/rescale) for this purpose, but the effects are not always satisfactory. If you are simulating a system that also contains non-rigid atoms (e.g. solvent), then you can thermostat those atoms and hope they will couple to the rigid bodies. The right solution is probably to enhance this fix to allow for direct temperature control of the rigid bodies.

Related commands:

delete_bonds, neigh_modify exclude

Default: none

[LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#)

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.

Restrictions: none

Related commands:

[fix addforce](#), [fix aveforce](#)

Default: none

[LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#)

fix shake style

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 4 a 3 5 2
fix 1 sub shake 0.0001 20 10 t 4 6 4 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. 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 virial is also accounted for.

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

[LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#)

fix spring style

Syntax:

```
fix ID group spring x y z k
```

- ID, group-ID are documented in [fix](#) command
- spring = style name of this fix command
- x,y,z = point to which spring is tethered
- k = spring constant (force/distance units)

Examples:

```
fix pull ligand spring 0.0 1.0 1.0 5.0
```

Description:

Apply a spring force to a group of atoms. An example is an umbrella force on an small molecule. A second example is 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.

Each timestep, the center of mass R of the group of atoms is computed, taking account of wrap-around in a periodic simulation box. A restoring force of magnitude $k (R - R_0) M_i / M$ is applied to each atom in the group where k is the specified force constant, R_0 is the tethering point of the spring specified by (x,y,z), 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.

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.

Restrictions: none

Related commands:

[fix drag](#)

Default: none

[LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#)

fix temp/rescale command

Syntax:

```
fix ID group-ID temp/rescale Tstart Tstop N window fraction
```

- ID, group-ID are documented in [fix](#) command
- temp/rescale = style name of this fix command
- Tstart,Tstop = desired temperature at start/end of run (temperature units)

- N = perform rescaling every N steps
- 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 1.0 1.1 100 0.02 0.5
```

Description:

Reset the temperature of a group of atoms by explicitly rescaling their velocities. The target temperature is a ramped value between the T_{start} and T_{stop} temperatures at the beginning and end of the run.

Rescaling is only performed every N timesteps, and only 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.

The way that temperature is computed by this fix can be changed by using the [fix_modify](#) command.

A temp/rescale fix does not update the coordinates of its atoms. It is normally used with a fix of style *nve* that does that. A temp/rescale fix should not normally be used on atoms that also have their temperature controlled by another fix – e.g. a [nvt](#) or [langevin](#) fix.

Restrictions: none

Related commands:

[fix_langevin](#), [fix_nvt](#), [fix_modify](#)

Default: none

[LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#)

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 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 [\(Schlitter2\)](#).

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.

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).

[LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#)

fix viscous command

Syntax:

```
fix ID group-ID viscous coeff
```

- ID, group-ID are documented in [fix](#) command
- viscous = style name of this fix command
- coeff = damping coefficient (unitless)

Examples:

```
fix 1 flow viscous 0.1
```

Description:

Add a viscous damping force to atoms in the group that is proportional to the velocity of the atom. This is useful for draining the kinetic energy from the system in a controlled fasion. The damping force F is given by $F = -\text{coeff} * \text{velocity}$. The larger the coefficient, the faster the kinetic energy is reduced.

Restrictions: none

Related commands: none

Default: none

[LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#)

fix volume/rescale command

Syntax:

```
fix ID group-ID volume/rescale N keyword args ...
```

- ID, group-ID are documented in [fix](#) command

- volume/rescale = style name of this fix command
- N = perform volume rescaling every this many timesteps
- one or more keyword/value pairs may be appended to the args
- keyword = *x* or *y* or *z*

x, y, z args = lo,hi = desired simulation box boundaries
at end of run

Examples:

```
fix 1 all volume/rescale x -9.0 9.0 z -5.0 5.0
```

Description:

Enable a volume (density) change during a simulation. Each of the 3 box dimensions is controlled separately. Any dimension being varied by this command must be periodic – see the [boundary](#) command. Dimensions not varied by this command can be periodic or non-periodic. The volume associated with an unspecified dimension can also be controlled by a [fix npt](#) command.

The initial simulation box boundaries at the beginning of a run are specified by the [create_box](#) or [read_data](#) or [read_restart](#) command used to setup the simulation, or they are the values at the end of the previous run. The desired simulation box boundaries at the end of the run are given by the *lo* and *hi* arguments. Every Nth timestep during the run, the simulation box is expanded or contracted to an ramped value between the initial and final values. The coordinates of all atoms in the group are also scaled to the new box size.

Restrictions:

Any dimension being varied by this fix must be periodic.

Related commands: none

Default: none

[LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#)

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 (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
values = dim amplitude period
dim = x or y or z
amplitude = size of oscillation (distance units)
period = time of oscillation (time 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, a wall can be oscillated, similar to the oscillations of frozen particles specified by the [fix_wiggle](#) command. This is useful in packing simulations of granular particles. If the keyword *wiggle* is appended to the argument list, then a dimension for the motion, as well as its *amplitude* and *period* is specified. Each timestep, the position of the wall in the appropriate *dim* is set according to this equation:

$$\text{position} = \text{pos0} + A - A \cos(\omega * \text{delta})$$

where *pos0* is the position at the time the fix was specified, *A* is the *amplitude*, ω is $2 \text{ PI} / \text{period}$, and *delta* is the elapsed time since the fix was specified. The velocity of the wall is also set to the derivative of this expression.

Restrictions:

A *zcylinder* wall can only be oscillated in the z dimension. This fix can only be used with *atom_style granular*.

Related commands:

[fix_wiggle](#)

Default: none

fix wall/93 command

Syntax:

```
fix ID group-ID wall/lj93 style coord epsilon sigma cutoff
```

- 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

Examples:

```
fix wallhi all wall/lj93 xhi 10.0 1.0 1.0 1.12
```

Description:

Bound the simulation domain with a Lennard-Jones wall that encloses the atoms. The energy E of a 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.

Restrictions: none

Related commands: none

Default: none

[LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#)

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 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 $2\pi / \text{period}$, and *delta* is the elapsed time since the fix was specified. The velocity of the atom is set to the derivative of this expression.

Restrictions: none

Related commands: none

Default: none

[LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#)

group command

Syntax:

```
group ID style args
```

- ID = user-defined name of the group
- style = *region* or *type* or *id* or *molecule* or *join* or *subtract*

```
region args = one or more region IDs
type or id or molecule
  args = one or more atom types, atom IDs, or molecule IDs
  args = logical value
    logical = "" 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)
join args = one or more group IDs
subtract args = two or more group IDs
```

Examples:

```
group edge region reg1 reg2
group water type 3 4
group sub id <= 150
group polyA molecule 50 250
```



```
group boundary join lower upper
group boundary subtract all a2 a3
```

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 listed regions into the group. An atom must be contained in each of the regions to qualify. If you want to add atoms to the same group that are in geometrically distinct regions, use the group command multiple times.

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 5 valid logicals are listed above. All the logicals except *between* 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 *join* style takes a list of one or more existing group names as arguments. All atoms that belong to any of these groups are added to the specified group.

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.

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.

[LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#)

improper_coeff command

Syntax:

```
improper_coeff N args
```

- N = improper type (see asterik 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 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 asterik 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 asterik with no numeric values means all types from 1 to N. A leading asterik means all types from 1 to n (inclusive). A trailing asterik means all types from n to N (inclusive). A middle asterik means all types from m to n (inclusive).

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 no wild-card asterik can be used. 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
```

See the improper_style command for the formulas that use these coefficients. The units of each coefficient are shown in parenthesis.

For style *harmonic*, specify 2 coefficients:

- K (energy)
- X0 (degrees)

For style *cvff*, specify 3 coefficients:

- K (energy)
- d (+1 or -1)
- n (0,1,2,3,4,6)

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:

Default: none

improper_style command

Syntax:

```
improper_style style
```

- style = *none* or *harmonic* or *cvff*

Examples:

```
improper_style harmonic  
improper_style cvff
```

Description:

Set the formula LAMMPS will use to compute improper interactions for quadruplets (trigonal centers) of atoms. The list of atom quadruplets are specified in the data or restart file and are read in by a [read_data](#) or [read_restart](#) command. The coefficients for the formula for each improper type can also be specified in those files or by the [improper_coeff](#) command.

A style of *none* means improper forces are not computed, even if impropers are defined.

The *harmonic* 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. K and χ_0 are coefficients defined for each improper type.

The *cvff* style uses the potential

$$E = K[1 + d \cos(n\phi)]$$

where ϕ is the Wilson out-of-plane angle. K , d , and n are coefficients defined for each improper type.

Restrictions:

Improper styles can only be set for atom_style choices that allow impropers to be defined.

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.

Related commands:

[improper_coeff](#)

Default:

`improper_style none`

[LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#)

jump command

Syntax:

```
jump file
```

- file = filename of new input script to switch to

Examples:

```
jump newfile  
jump in.run2
```

Description:

This command closes the current input script file, opens the file with the specified name, and begins reading LAMMPS commands from that file. The original file is not returned to.

It is possible to chain from file to file or back to the original file using successive jump commands. If the filename is a variable (see the [variable](#) command), different processor partitions can run different input scripts.

Restrictions: none

Related commands:

[variable](#)

Default: none

[LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#)

kspace_modify command

Syntax:

```
kspace_modify keyword value ...
```

- one or more keyword/value pairs may be listed
- keyword = *mesh* or *order* 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
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 *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, and slab = 1.0.

[LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#)

kspace_style command

Syntax:

```
kspace_style style value
```

kspace_style command

- style = *none* or *ewald* or *pppm* none value = none *ewald* value = precision precision = desired accuracy *pppm* 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. When such a solver is used in conjunction with an appropriate pair style, the cutoff for Coulombic 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 an 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](#)).

When a kspace style is used, a pair style that includes the short-range correction to the pairwise Coulombic forces must also be selected. These styles are *lj/cut/coul/long* and *lj/charmm/coul/long*.

A precision value of 1.0e-4 means one part in 10000. This setting is used in conjunction with the pairwise cutoff to determine the number of K-space vectors for style *ewald* or the FFT grid size for style *pppm*.

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.

Related commands:

[kspace_modify](#), [pair_style](#) *lj/cut/coul/long*, [pair_style](#) *lj/charmm/coul/long*

Default:

```
kspace_style none
```

(**Darden**) Darden, York, Pedersen, J Chem Phys, 98, 10089 (1993).

(**Hockney**) Hockney and Eastwood, Computer Simulation Using Particles, Adam Hilger, NY (1989).

lattice command

Syntax:

```
lattice style value
```

- style = *none* or *sc* or *bcc* or *fcc* or *sq* or *sq2* or *hex*

```
none value = none
for all other styles:
  value = reduced density (for LJ units)
  value = cubic lattice constant in Angstroms (for real or metal units)
```

Examples:

```
lattice fcc 3.52
lattice hex 0.85
lattice none
```

Description:

Define a lattice type and lattice constant. This is required before using a commands that (optionally) use the lattice, such as [create_atoms](#) or [region](#). The lattice type must be consistent with the dimension of the simulation – see the [dimension](#) command. Styles *sc* or *bcc* or *fcc* are for 3d problems. Styles *sq* or *sq2* or *hex* are for 2d problems. Lattices of style *fcc*, *bcc*, *sc*, or *hex* are described in any solid–state physics text. A *sq* lattice is one with atoms at the corners of a square. A *sq2* lattice is a *sq* lattice with an additional atom at the center of the square.

For unit style *real* or *metal*, the specified value is the cubic lattice constant in Angstroms. For unit style *lj*, the value is the reduced density (ρ^*) which LAMMPS converts into a cubic lattice constant. For 3d problems, the relationship " $\rho^* = \rho \sigma^3$ " is used for the conversion, where $\rho = N/V$ with V = the volume of the cubic cell and $N = 4$ for *fcc*, 2 for *bcc*, and 1 for *sc* (simple cubic) lattices. For 2d problems, the relationship " $\rho^* = \rho \sigma^2$ " is used for the conversion, where $N = 2$ for *sq2* or *hex* and 1 for *sq*. In the hex case, the unit cell is actually rectangular; it is extended by a factor of $\sqrt{3}$ in the y–dimension.

The command "lattice none" can be used to turn off the lattice setting. Any command that attempts to use a lattice constant will then generate an error.

Restrictions: none

Related commands:

[dimension](#), [orient](#), [origin](#), [create_atoms](#), [region](#)

Default:

```
lattice none
```

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 asterik 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.

It 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 asterik 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 asterik with no numeric values means all types from 1 to N. A leading asterik means all types from 1 to n (inclusive). A trailing asterik means all types from n to N (inclusive). A middle asterik means all types from m to n (inclusive).

A line in a data file that specifies mass uses the exact same format as the arguments of the mass command in an input script, except that no wild-card asterik 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.

Masses are not set for atom style granular. This is because each atom is assigned an individual mass in the data or restart file.

Related commands: none

Default: none

[LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#)

neigh_modify command

Syntax:

```
neigh_modify keyword values ...
```

- one or more keyword/value pairs may be listed

```
keyword = delay or every or check or exclude
delay value = N
    N = delay building until this many steps since last build
every value = N
    N = 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
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

```

Examples:

```

neigh_modify every 2 delay 5 check yes
neigh_modify exclude type 2 3
neigh_modify exclude group frozen frozen check no
neigh_modify exclude group residue1 chain3
neigh_modify exclude molecule rigid

```

Description:

This command sets parameters that affect the pairwise neighbor list.

The *every*, *delay*, and *check* options affect how often the list is 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 N 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 *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.

Restrictions:

The exclude molecule option can only be used with atom styles that define molecule IDs.

Related commands:

[neighbor, delete_bonds](#)

Default:

The option defaults are delay = 10, every = 1, check = yes, and exclude = none.

[LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#)

neighbor command

Syntax:

```
neighbor skin style
```

- skin = extra distance beyond force cutoff (distance units)
- style = *bin* or *nsq*

Examples:

```
neighbor 0.3 bin  
neighbor 2.0 nsq
```

Description:

This command sets parameters that affect the building of the pairwise neighbor list. All atom pairs within a 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 *style* value chooses what algorithm is used to build the list. *Binning* is an operation that scales linearly with N, the number of atoms on a processor. It is almost always faster than the *nsq* style which scales as N^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 default values for *skin* and *style* depend on the choice of units for the simulation.

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](#)

Default:

```
0.3 bin      for lj units
2.0 bin      for real or metal units
```

[LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#)

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
```

newton command

next command

Syntax:

```
next variables
```

- variables = one or more lower-case single-character variable names

Examples:

```
next x
next a t x
```

Description:

This command is used with variables of the *index* style as defined by the [variable](#) command. It sets the value of the variable to the next argument in the variable list, so that when \$X is substituted for in an input script line, the new value is used. X is a single lower-case character from "a" to "z".

The behavior of the next command depends on whether LAMMPS is running on a single partition or multiple partitions. See [this section](#) for a discussion of the `-partition` command-line switch.

If all processors are in one partition, then occurrences of \$X are substituted for on all processors. Initially the variable is set to the 1st argument. After a [next](#) command is invoked for variable X, the 2nd argument is used in substitutions for \$X, etc. For example, an input script `in.polymer` could use these commands to run a series of 8 simulations from directories `run1` thru `run8`.

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

When the [next](#) command increments an *index* style variable past its final value, LAMMPS exits. This prevents scripts, like the example just given, from looping endlessly.

When LAMMPS is running on multiple partitions, a variable command for an *index* style variable assigns a different value to each partition. I.e. on 4 partitions, the first 4 values of the variable are assigned, one to each partition. When a *next* command is encountered, the first partition to invoke the command assigns the next available argument to the variable. A partition which invokes the *next* command later in time will assign the next available value, etc. If you have several variables that must be incremented simultaneously in this fashion, list them as arguments to a single *next* command.

On 3 partitions, the same `in.polymer` script above would run the 8 simulations on 3 sets of processors. Whenever a partition finishes one simulation it will set the variable *d* to the next available value and run another simulation. When all 8 simulations finish on whatever partitions they ran on, LAMMPS will exit.

To coordinate the variable usage between multiple partitions, LAMMPS creates a temporary file, `lammps.variable`, with information about variable status. This file should not be deleted or modified while LAMMPS is running.

Restrictions: none

Related commands:

[variable](#), [jump](#)

Default: none

[LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#)

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_coeff command

Syntax:

```
pair_coeff I J args
```

- I,J = atom types (see asterik form below)
- args = coefficients for one or more pairs of atom types

Examples:

```
pair_coeff 2 2 1.0 1.0 2.5
pair_coeff 2 * 1.0 1.0 2.5
pair_coeff 3* 1*2 1.0 1.0 2.5
pair_coeff * * 1.0 1.0 2.5
pair_coeff 2 2 niu3
```

```
pair_coeff * * nialhjea 1 1 2
```

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 read_data command or in a restart file.

The one exception is that the information needed for pair style *eam* can only be set by the `pair_coeff` command. Similarly, coefficient data for pair style *eam* is not written to restart files (see the restart command), so a `pair_coeff` command must be used in the input script that reads the restart file to setup EAM potentials.

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. In this case, $I \leq J$ is required. LAMMPS sets the coefficients for the symmetric J,I interaction to the same values.

A wild-card asterik can be used 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 asterik with no numeric values means all types from 1 to N. A leading asterik means all types from 1 to n (inclusive). A trailing asterik means all types from n to N (inclusive). A middle asterik means all types from m to n (inclusive). Note that only type pairs with $I \leq J$ are set; if asteriks imply type pairs where $J < I$, they are ignored.

Note that using 2 `pair_coeff` commands for the same I,J pair is perfectly valid. For example, these two 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 2 exceptions. In a data file, wild-card asteriks cannot be used and I,J pairs are not specified. Only a single type I is specified, which sets the coefficients for type I interacting with type I. This is because in a data file, the "Pair Coeffs" section has exactly N lines, where N = the number of atom types. The coefficients for type pairs with I not equal J are inferred from the N settings by mixing rules; see the pair_modify command for a discussion. Thus in a data file, the line corresponding to the 1st example above would be listed as

```
2 1.0 1.0 2.5
```

Here are the coefficients specified by the `pair_coeff` command for each pair style. Note that any cutoffs specified by these coefficients for a particular I,J pair override the global cutoff(s) specified by the pair_style command.

See the pair_style command for the formulas that use these coefficients.

For style *lj/cut*, specify 3 coefficients:

- epsilon (energy units)
- sigma (distance units)
- cutoff (distance units)

For style *lj/cut/coul/cut*, specify 3 or 4 coefficients:

- epsilon (energy units)
- sigma (distance units)
- cutoff1 (distance units)
- cutoff2 (distance units)

The 2nd cutoff is optional. If it is not specified, cutoff1 = the LJ and Coulombic cutoff. If it is specified, cutoff1 = LJ cutoff and cutoff2 = Coulombic cutoff.

For style *lj/cut/coul/long*, specify 3 coefficients:

- epsilon (energy units)
- sigma (distance units)
- cutoff (distance units)

The cutoff is the LJ cutoff. You cannot specify a Coulombic cutoff for an individual I,J type pair. All type pairs use the same global Coulombic cutoff specified in the *pair_style* command.

For style *lj/expand*, specify 4 coefficients:

- epsilon (energy units)
- sigma (distance units)
- delta (distance units)
- cutoff (distance units)

The delta values can be positive or negative. Note that the cutoff does not include the delta distance. I.e. the actual force cutoff is the sum of cutoff + delta.

For style *soft*, specify 3 coefficients:

- 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.

For styles *lj/charmm/coul/charmm*, *lj/charmm/coul/charmm/implicit*, and *lj/charmm/coul/long*, specify 4 coefficients:

- epsilon (energy units)
- sigma (distance units)
- epsilon_14 (energy units)
- sigma_14 (distance units)

The latter 2 coefficients are optional. If they are specified, they are used in the Lennard–Jones formula between 2 atoms of these types that are first and fourth atoms in any dihedral. No cutoffs are specified because this CHARMM force field does not allow cutoffs for individual atom pairs; all pairs use the global cutoff(s) specified in the *pair_style* command.

For style *eam*, coefficients are listed in one of two forms. For single–element DYNAMO potential files, specify 1 coefficient:

- filename

The EAM fits for F, rho, phi (see the pair_style command) are read from the file and assigned to atom type(s) specified by the first two pair_coeff command arguments, for type $i = j$ (see below for mixing discussion). Use the pair_coeff command as many times as necessary to assign filenames (EAM elements) to each atom type in the LAMMPS simulation.

For multi–element DYNAMO potential files, the file contains information for M elements. These have to be mapped to LAMMPS atom types, which is done by specifying N+1 coefficients, where N is the number of atom types:

- filename
- N values from 1 to M = mapping of setfl elements to atom types

For example, the *nialhjea* setfl file has EAM data for 3 elements (and their alloy interactions) – Ni, Al, and H. If your LAMMPS simulation has 4 atoms types and you want the 1st 3 to be Ni, and the 4th to be Al. Then you would use the pair_coeff command:

```
pair_coeff * * nialhjea 1 1 1 2
```

The 3 "1" values map LAMMPS atom types 1,2,3 to the 1st element (Ni) in the setfl file. The "2" value maps atom type 4 to Al. Note that you must use "*" to cover all atom types I,J in the pair_coeff command. And you only use the pair_coeff command once to specify EAM force fields for all atom types in the LAMMPS simulation.

As these instructions imply, you cannot mix funcfl and setfl DYNAMO files in the same simulation. EAM potentials are "mixed" (type I interactions with type J) by their own rules; you do not need to specify $I \neq J$ interactions or use the pair_modify command's *mix* option.

The formats of the single–element (funcfl) and multi–element (setfl) DYNAMO files are beyond the scope of this manual. You'll have to look in the *src/pair_eam.cpp* file and/or the DYNAMO files themselves to figure out the format. You don't need to know what's in these files to use them, and if you intend to derive your own potentials for new materials, you'll presumably know enough about EAM potentials and DYNAMO to get started.

For styles *gran/no_history*, *gran/history*, and *gran/hertzian*, there are no individual atom type coefficients that can be set. All global settings are made via the pair_style command.

For style *lj/cut/coul/debye*, the 3 or 4 coefficients are the same as for style *lj/cut/coul/cut*. The kappa value specified in the pair_style command is global and is not set for individual atom type pairs.

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](#)

Default: none

[LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#)

pair_modify command

Syntax:

```
pair_modify keyword value ...
```

- one or more keyword/value pairs may be listed
- keyword = *shift* or *mix*

```
shift value = yes or no
mix value = geometric or arithmetic or sixthpower
table value = N
2^N = # of values in table
```

Examples:

```
pair_modify shift yes
pair_modify mix arithmetic
pair_modify table 12
```

Description:

Modify the parameters of the currently defined pair style. Not all parameters are relevant to all pair styles.

The *shift* keyword determines whether the 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 printed in the thermodynamic output, but does not affect atom dynamics (forces). Pair styles that are already 0.0 at their cutoff such as *lj/charmm/coul/charmm* are not affected by this setting.

The *mix* keyword affects how Lennard–Jones coefficients for epsilon and sigma are generated for interactions between atoms of type I and J. Note that the epsilon/sigma for I = J are set explicitly in the data file or input script. The [pair_coeff](#) command can also be used in the input script to specify epsilon/sigma for a specific I,J pairing, which overrides the setting of the *mix* keyword.

These are the formulas used by the 3 *mix* options:

geometric

```
epsilon_ij = sqrt(epsilon_i * epsilon_j)
sigma_ij = sqrt(sigma_i * sigma_j)
```

arithmetic

```
epsilon_ij = sqrt(epsilon_i * epsilon_j)
sigma_ij = (sigma_i + sigma_j) / 2
```

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)
```

Style *soft* only uses a pre-factor coefficient, which is always mixed geometrically, regardless of the *mix* setting. Styles *lj/charmm/coul/charmm* and *lj/charmm/coul/long* are always mixed arithmetically, regardless of the *mix* setting. Style *lj/expand* always mixes its delta coefficient using the rule

```
delta_ij = (delta_i + delta_j) / 2
```

The *table* keyword applies to pair styles with a long-range Coulombic term (*lj/cut/coul/long* and *lj/charmm/coul/long*). 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 table is only used for pair distances >= 2.0 which means this option will have more effect on simulations in "real" units than those in "lj" units.

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.

Restrictions: none

Related commands:

[pair_style](#), [pair_coeff](#)

Default:

The option defaults are shift = no, mix = arithmetic (for *lj/charmm* pair styles), mix = geometric (for other pair styles), and table = 12.

([Wolff](#)) Wolff and Rudd, Comp Phys Comm, 120, 200–32 (1999).

[LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#)

pair_style command

Syntax:

```
pair_style style args
```

- *style* = *none* or *lj/cut* or *lj/cut/coul/cut* or *lj/cut/coul/long* or *lj/expand* or *soft* or *lj/charmm/coul/charmm* or *lj/charmm/coul/charmm/implicit* or *lj/charmm/coul/long* or *eam* or *gran/no_history* or *gran/history* or *gran/hertzian* or *lj/cut/coul/debye*

```

none args = none
  lj/cut args = cutoff
    cutoff = force cutoff distance for Lennard-Jones interactions
  lj/cut/coul/cut args = cutoff
    cutoff = force cutoff distance for LJ and Coulombic interactions
  lj/cut/coul/cut args = cut1, cut2
    cut1 = cutoff for LJ
    cut2 = cutoff for Coulmbic
  lj/cut/coul/long args = cutoff
    cutoff = cutoff for LJ and Coulombic
  lj/cut/coul/long args = cut1, cut2
    cut1 = cutoff for LJ
    cut2 = cutoff for Coulmbic
  lj/expand args = cutoff
    cutoff = cutoff for LJ
soft = cutoff
  cutoff = cutoff for soft interactions
lj/charmm/coul/charmm args = inner outer
  inner, outer = switching cutoffs for LJ and Coulmbic
lj/charmm/coul/charmm args = inner1 outer1 inner2 outer2
  inner1, outer1 = switching cutoffs for LJ
  inner2, outer2 = switching cutoffs for Coulombic
lj/charmm/coul/charmm/implicit args = inner outer
  inner, outer = switching cutoffs for LJ and Coulmbic
lj/charmm/coul/charmm/implicit args = inner1 outer1 inner2 outer2
  inner1, outer1 = switching cutoffs for LJ
  inner2, outer2 = switching cutoffs for Coulombic
lj/charmm/coul/long args = inner outer
  inner, outer = switching cutoffs for LJ
  outer = cutoff for Coulombic
lj/charmm/coul/long args = inner outer cutoff
  inner, outer = switching cutoffs for LJ
  cutoff = cutoff for Coulombic
eam args = none
gran/no_history args = xkk, gamman, xmu
gran/history args = xkk, gamman, xmu
gran/hertzian args = xkk, gamman, xmu
  xkk = spring constant for particle overlap repulsion
  gamman = damping coefficient for normal direction collisions
  xmu = friction coefficient
lj/cut/coul/debye args = cutoff kappa
  cutoff = cutoff for LJ and Coulmbic
  kappa = Debye length (distance units)
lj/cut/coul/debye args = cut1 cut2 kappa
  cut1 = cutoff for LJ
  cut2 = cutoff for Coulombic
  kappa = Debye length (inverse distance units)

```

Examples:

```

pair_style none
pair_style lj/cut 2.5
pair_style lj/cut/coul/cut 2.5
pair_style lj/cut/coul/cut 10.0 8.0
pair_style lj/cut/coul/long 12.0
pair_style lj/expand 2.5
pair_style soft 2.0

```

pair_style command

```
pair_style lj/charmm/coul/charmm 10.0 10.3
pair_style lj/charmm/coul/charmm 10.0 10.3 8.0 8.3
pair_style lj/charmm/coul/charmm/implicit 10.0 10.3
pair_style lj/charmm/coul/long 10.0 10.3
pair_style eam
pair_style gran/history 200000.0 0.5 1.0
```

```
pair_style lj/cut/coul/debye 2.5 3.0
```

Description:

Set the style of interactions for computing pairwise forces and energies. In LAMMPS, pairwise computations include both van der Waals (Lennard Jones) and Coulombic interactions, so there is a range of style choices that encompass the combination of both kinds of forces.

The coefficients for pairwise interactions are set by the pair_coeff command or read from a file by the read_data or read_restart commands. Mixing and shifting of the interaction potentials is discussed with the pair_modify command.

The cutoff arguments set global cutoffs for all type atom pairs which can be overridden by the pair_coeff command. The global cutoff can also be changed by a subsequent pair_style command using the same style. This will only change cutoffs for atom type pairs that have not been explicitly set by the pair_coeff command.

All cutoff arguments are in distance units. The specified cutoff distance(s) can be smaller or larger than the dimensions of the simulation box.

In the formulas to follow, E is the energy of a pairwise interaction between two atoms separated by a distance r . The force between the atoms is the derivative of this expression.

Style *none* turns off pairwise interactions.

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.

Style *lj/cut* is a cutoff Lennard–Jones (LJ) potential, given by

$$E = 4\epsilon \left[\left(\frac{\sigma}{r} \right)^{12} - \left(\frac{\sigma}{r} \right)^6 \right] \quad r < r_c$$

where epsilon and sigma are coefficients defined for each pair of atom types. r_c is the cutoff.

Style *lj/cut/coul/cut* computes the same LJ formula as style *lj/cut*. Additionally, it computes Coulombic pairwise interactions as

$$E = \frac{Cq_iq_j}{\epsilon r} \quad r < r_c$$

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 or pair_coeff 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/long* computes the same pairwise 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 these styles means that pairwise interactions within this distance are computed directly; interactions outside that distance are computed in K-space.

Style *lj/expand* computes a LJ interaction with a distance shifted by delta

$$E = 4\epsilon \left[\left(\frac{\sigma}{r - \Delta} \right)^{12} - \left(\frac{\sigma}{r - \Delta} \right)^6 \right] \quad r < r_c + \Delta$$

The epsilon, sigma, and delta coefficients are defined for each pair of atom types. Rc is the cutoff.

Style *soft* is useful for pushing apart overlapping atoms, since it does not blow up as r goes to 0. It computes a pairwise interaction as

$$E = A \left[1 + \cos \left(\frac{\pi r}{r_c} \right) \right] \quad r < r_c$$

where A is a pre-factor that varies in time from the start to the end of the run. Starting and ending values for A are specified by the pair_coeff or read_data command. Rc is the cutoff.

Style *lj/charmm/coul/charmm* computes 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 option in the CHARMM MD code.

$$\begin{aligned}
E &= LJ(r) & r < r_{in} \\
&= S(r) * LJ(r) & r_{in} < r < r_{out} \\
&= 0 & r > r_{out} \\
E &= C(r) & r < r_{in} \\
&= S(r) * C(r) & r_{in} < r < r_{out} \\
&= 0 & r > r_{out} \\
LJ(r) &= 4\epsilon \left[\left(\frac{\sigma}{r} \right)^{12} - \left(\frac{\sigma}{r} \right)^6 \right] \\
C(r) &= \frac{C q_i q_j}{\epsilon r} \\
S(r) &= \frac{[r_{out}^2 - r^2]^2 [r_{out}^2 + 2r^2 - 3r_{in}^2]}{[r_{out}^2 - r_{in}^2]^3}
\end{aligned}$$

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 typically 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 style `lj/cut/coul/long`.

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

$$E_i = F \left(\sum_{j \neq i} \rho(r_{ij}) \right) + \sum_{j \neq i} \phi(r_{ij})$$

where F is the embedding energy which is a function of the atomic electron density ρ , and ϕ is a pair potential interaction. The multi-body nature of the EAM potential is a result of the embedding energy term. Both summations in the formula are over all neighbors J of atom I within the cutoff distance.

The cutoff distance and the numeric form of F , ρ , and ϕ are listed in one or more files which are specified by the `pair_coeff` command. Several files for different metals are in the "potentials" directory of the LAMMPS distribution. These are ASCII text files in a DYNAMO-style format. DYNAMO was a serial MD code authored by two of the EAM originators, Stephen Foiles and Murray Daw.

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 .

$$F = f(\delta/d) (k_n \delta \mathbf{n}_{ij} - m_{\text{eff}} \gamma_n \mathbf{v}_n) + f(\delta/d) (-k_t \delta \Delta \mathbf{s}_t - m_{\text{eff}} \gamma_t \mathbf{v}_t)$$

The 1st term is a normal force and the 2nd term is a tangential force. The other quantities are $\delta = d - r$, $f(x) = 1$ for Hookean contacts used in the pair styles *history* and *no_history*, $f(x) = \sqrt{x}$ for pair style *hertzian*, K_n and K_t = elastic constants for normal and tangential contact, γ_n and γ_t = viscoelastic constants for normal and tangential contact, $m_{\text{eff}} = M_i M_j / (M_i + M_j)$ = effective mass between 2 particles of mass M_i and M_j , $\Delta \mathbf{s}_t$ = the tangential displacement vector between the 2 spherical particles which is truncated to satisfy a frictional yield criterion, \mathbf{n} = a unit vector along the line connecting the centers of the 2 particles, and \mathbf{v}_n and \mathbf{v}_t = normal and tangential components of the relative velocity of the 2 particles. See the citation for more discussion of the granular potentials.

Style *lj/charmm/coul/debye* computes the same LJ formula as style *lj/cut*, but the Coulomb formula has an additional term

$$E = \frac{C q_i q_j}{\epsilon r} \exp(-\kappa r) \quad r < r_c$$

where κ is the Debye length. This potential is another way to mimic the screening effect of a polar solvent.

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.

The *gran/hertzian*, *gran/history*, and *gran/no_history* styles are part of the "granular" package. The *lj/charmm/coul/charmm* and *lj/charmm/coul/charmm/implicit* styles are part of the "molecule" package. The *lj/cut/coul/long* and *lj/charmm/coul/long* styles are part of the "k-space" package. The *eam* style is part of the "metal" package.

Related commands:

pair_coeff, read_data, pair_modify, k-space style, dielectric

Default:

```
pair_style none
```

(Daw) Daw, Baskes, Phys Rev Lett, 50, 1285 (1983). Daw, Baskes, Phys Rev B, 29, 6443 (1984).

(Silbert) Silbert, Ertas, Grest, Halsey, Levine, Plimpton, Phys Rev E, 64, p 051302 (2001).

LAMMPS WWW Site – LAMMPS Documentation – LAMMPS Commands

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

[LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#)

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 skipped; it typically contains a description of the file. Then lines are read one at a time. If the line is blank (all whitespace), 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) is read from the beginning of the line. Thus the keyword *atoms* should be in a line like "1000 atoms" and the keyword *ylo yhi* should be in a line like "-10.0 10.0 ylo yhi". 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
- *xlo xhi* = simulation box boundaries in x dimension
- *ylo yhi* = simulation box boundaries in y dimension
- *zlo zhi* = simulation box boundaries in z dimension

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.

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 system is non-periodic (in a dimension), then all atoms in the data file should have coordinates (in that dimension) between the lo and hi values. 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.

These are the section keywords for the body of the file.

- *Atoms, Velocities, Masses, Dipoles* = atom-property sections
- *Bonds, Angles, Dihedrals, Impropers* = molecular topology sections
- *Pair Coeffs, Bond Coeffs, Angle Coeffs, Dihedral Coeffs, & Improper Coeffs* = force field sections

The format of each section is now described including the number of lines it must contain and rules (if any) for where it can appear in the data file.

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.

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.

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.

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.

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.

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.

Structure of *Improper Coeffs* section:

processors command

- one line per improper type
- line syntax: ID coeffs

```
ID = improper type (1-N)
coeffs = list of coeffs
```

- example:

```
2 20 0.0548311
```

The number and meaning of the coefficients are specific to the defined improper style. See the [improper_style](#) and [improper_coeff](#) commands for details. Coefficients can also be set via the [improper_coeff](#) command in the input script.

Atoms section:

- one line per atom
- line syntax: depends on atom style

The format for the various atom styles is as follows:

- atomic = atom-ID type-ID x y z
- bond = atom-ID mol-ID type-ID x y z
- angle = atom-ID mol-ID type-ID x y z
- full = atom-ID mol-ID type-ID q x y z
- eam = atom-ID type-ID x y z
- granular = atom-ID type-ID diameter density x y z
- atom-ID = unique integer ID of the atom (1-N)
- type-ID = integer type ID of the atom (1-N)
- x,y,z = coordinates of the atom
- mol-ID = integer molecule ID of the molecule the atom belongs to
- q = charge on the atom
- diameter = diameter of atom
- density = density of atom

The units for these quantities depend on the unit style; see the [units](#) command for details. Note that all ID values must be specified as integers (1, not 1.0).

For 2d simulations specify z as 0.0, or whatever value is 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.

The molecule ID is a 2nd identifier attached to the atom. It can be 0 if it is an unbonded atom or if you don't wish to make molecule assignments.

An *Atoms* section must appear in the data file if *natoms* > 0 in the header section. The atoms can be listed in any order.

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. An image of 0 means 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 via the dump and dump modify commands. If nx,ny,nz values are not set in the data file, LAMMPS initializes them to 0.

Atom velocities are set to 0.0 when the *Atoms* section is read. They may later be set by a *Velocities* section or by a velocity command in the input script.

Velocities section:

- one line per atom
 - line syntax: atom-ID vx vy vz
- atom-ID = atom ID (1-N)
vx,vy,vz = components of velocity of the atom
- example:
- ```
45 -3.4 0.05 1.25
```

The velocity lines can appear in any order. This section can only be used after an *Atoms* section. Velocities can also be set by the velocity command in the input script.

*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).

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. 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).

*Dihedrals* section:

- one line per dihedral
- line syntax: ID type atom1 atom2 atom3 atom4

ID = number of dihedral (1-Ndihedrals)  
 type = dihedral type (1-Ndihedraltypes)  
 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).

*Impropers* section:

- one line per improper
- line syntax: ID type atom1 atom2 atom3 atom4

ID = number of improper (1-Nimpropers)  
 type = improper type (1-Nimproprtypes)  
 atom1,atom2,atom3,atom4 = IDs of 1st,2nd,3rd,4th atoms in improper

- example:

```
12 3 17 29 13 100
```

The *Impropers* section must appear after the *Atoms* section. All values in this section must be integers (1, not 1.0).

**Restrictions:** none

**Related commands:**

[read\\_restart](#), [create\\_atoms](#)

**Default:** none

[LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#)

---

## read\_restart command

**Syntax:**

```
read_restart file
```

- file = name of binary restart file to read in

**Examples:**

```
read_restart save.10000
```

**Description:**



Read in a previously saved problem from a restart file. This allows continuation of a previous run.

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 the shake and langevin styles. 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.

A restart file stores the units and atom style, simulation box attributes, individual atoms and their attributes including molecular topology, force field styles and coefficients, special\_bonds settings, and atom group definitions. This means that commands for these quantities do not need to be specified in your input script that reads the restart file.

One exception is that pair\_style eam does not store its coefficient data in the restart file. Thus you need to use the pair\_coeff command to read the appropriate potential files after the restart file is read.

Another exception is that information about the kspace\_style setting is not stored in the restart file. Hence if you wish to invoke an Ewald or PPPM solver, this command must be re-issued after the restart file is read.

The restart file also stores values for any fixes that require state information to enable restarting where they left off. These include the nvt and npt styles that have a global state, as well as the msd and wall/gran styles that store information about each atom.

Fix commands are not stored in the restart file which means they must be specified in the input script that reads the restart file. To re-enable a fix whose state was stored in the restart file, the fix command in the new input script must use the same fix-ID and group-ID as the input script that wrote the restart file. LAMMPS will print a message indicating that the fix is being re-enabled.

Note that no other information is stored in the restart file. This means that your new input script should specify settings for quantities like timestep size, thermodynamic and dump output, etc.

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.

**Restrictions:** none

**Related commands:**

read\_data, write\_restart, restart

**Default:** none

read\_restart command

## region command

### Syntax:

```
region ID style args keyword value ...
```

- ID = user-assigned name for the region
- style = *block* or *sphere* or *cylinder*

```
rectangle args = xlo xhi ylo yhi zlo zhi
 xlo,xhi,ylo,yhi,zlo,zhi = bounds of block in all
 dimensions (distance units)
sphere args = x y z radius
 x,y,z = center of sphere (distance units)
 radius = radius of sphere (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)
```

- zero or more keyword/value pairs may be appended to the args
- 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
```

### 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 *rectangle* or *cylinder* styles can be specified as INF which means they extend all the way to the global simulation box boundary. If a region is defined before the simulation box has been created (via [create\\_box](#) or [read\\_data](#) or [read\\_restart](#) commands), then an INF parameter cannot be used.

For region *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.

The *side* keyword determines whether the region is considered to be inside or outside of the specified geometry.

The *units* keyword determines the meaning of the distance units used to defined 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 cubic lattice spacings. See the lattice must first be used to define a lattice.

**Restrictions:** none

**Related commands:**

lattice, create\_atoms, delete\_atoms, group

**Default:**

The option defaults are side = in and units = lattice.

[LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#)

---

## 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 (tags) 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.

This command operates similar to the replicate tool in the tools sub-directory of the LAMMPS distribution which creates new data files from old ones.

**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.

**Related commands:** none

**Default:** none

[LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#)

---

## 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

**Related commands:** none

**Default:** none

[LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#)

---

## 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 10000 poly.r.1 poly.r.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 with a timestep suffix, e.g. the 2nd example above will create poly.restart.1000, poly.restart.2000, poly.restart.3000, etc. Using two filenames will produce only 2 restart files. LAMMPS will toggle between the 2 names as it writes successive restart files.

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.

**Restrictions:** none

### Related commands:

[write\\_restart](#), [read\\_restart](#)

### Default:

```
restart 0
```

[LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#)

---

## run command

### Syntax:

```
run N
```

- N = # of timesteps

### Examples:

```
run 10000
```

### Description:

Run or continue dynamics for the 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.

**Restrictions:** none

**Related commands:**

[run\\_style](#), [temper](#)

**Default:** none

[LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#)

---

## 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 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.

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).

[LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#)

---

## set\_type command

**Syntax:**

```
set_type group-ID style N
```



- group-ID = ID of a group of atoms
- style = *atom* or *bond* or *angle* or *dihedral* or *improper*
- N = new type to change to (from 1 to # of types)

### Examples:

```
set_type solvent atom 2
set_type edge bond 4
```

### Description:

Change the atom type (bond type, angle type, etc) for all atoms (bonds, angles, etc) in a particular group. Since types are specified when atoms are created or read in from a data or restart file, this command changes those assignments. This can be useful for altering pairwise and molecular force interactions, since force-field coefficients are defined in terms of types. It can also be used to change the labeling of atoms when they are output in dump files.

For style *atom*, the atom type of all atoms in the group is changed to the specified value from 1 to ntypes. For style *bond* (*angle*, *dihedral*, *improper*), the bond type (angle type, etc) of each bond (angle, etc) in the group is changed to the specified value from 1 to nbondtypes (angletypes, etc). All atoms in the bond (angle, etc) must be in the group in order for the change to be made.

Note that new type values must be within the range the simulation was initialized for. In the case of atom types, the maximum number of types is set by the create\_box command or the *atom types* field in the header of the data file read by the read\_data command. In the case of bond types (angle types, etc), the maximum number of types is set by the *bond types* (*angle types*, etc) field in the header of the data file.

### Restrictions:

This command requires inter-processor communication to coordinate the setting of bond types (angle types, etc). This means that pairwise force cutoffs must be already be set before using this command, so that each processor can acquire the correct atoms. This is not necessary to reset atom types.

### Related commands:

create\_box, read\_data

**Default:** none

[LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#)

---

## special\_bonds command

### Syntax:

```
special_bonds style
special_bonds c1 c2 c3
special_bonds c1 c2 c3 c4 c5 c6
```

- style = *charmm* or *amber*
- c1,c2,c3,c4,c5,c6 = numeric coefficients from 0.0 to 1.0

Examples:

```
special_bonds charmm
special_bonds amber
special_bonds 0.0 0.0 1.0
special_bonds 0.0 0.0 1.0 0.0 0.0 0.5
```

### Description:

Set the weighting coefficients for the pairwise force and energy contributions from atom pairs that are also bonded to each other directly or indirectly. 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 2 H atoms in a water molecule). The 3rd coefficient is the weighting factor on 1–4 atom pairs which are separated by 3 bonds (e.g. the 1st and 4th atoms in a dihedral interaction).

1–3, and 1–4 interactions are not computed using the list of angles and dihedrals defined in the simulation. Rather, they are inferred by the set of defined bonds. This distinction is important to remember if bonds are removed at some point during a simulation. Also note that turning off a bond (as opposed to removing it) does not change the inference of 1–2, 1–3, and 1–4 neighbors. See the [delete\\_bonds](#) command for more details.

The *charmm* style sets all 3 coefficients to 0.0, which is the default for the CHARMM force field. In pair styles *lj/charmm/coul/charmm* and *lj/charmm/coul/long* the 1–4 coefficients are defined explicitly, and these pair-wise contributions are computed in the charmm dihedral style – see the [pair\\_coeff](#) and [dihedral\\_style](#) commands for more information.

The *amber* style sets the 3 coefficients to 0.0 0.0 0.5 for LJ interactions and to 0.0 0.0 0.833 for Coulombic interactions, which is the default for a particular version of the AMBER force field, where the last value is 5/6.

A `special_bonds` command with 3 coefficients sets the 1–2, 1–3, and 1–4 coefficients for both LJ and Coulombic terms to those values.

A `special_bonds` command with 6 coefficients sets the 1–2, 1–3, and 1–4 LJ coefficients to the first 3 values and the Coulombic coefficients to the last 3 values.

**Restrictions:** none

**Related commands:**

[delete\\_bonds](#)

**Default:**

```
special_bonds 0.0 0.0 0.0
```

---

[LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#)

---

## temp\_modify command

### Syntax:

```
temp_modify temp-ID keyword value ...
```

- temp-ID = ID of temperature to modify
- one or more keyword/value pairs may be listed
- keyword = *extra*

```
extra value = N
```

```
N = # of extra degrees of freedom to subtract
```

### Examples:

```
temp_modify mine extra 0
```

### Description:

Modify the parameters of a previously defined temperature command.

The *extra* keyword refers to how many degrees-of-freedom are subtracted from  $3N$  as a normalizing factor in the temperature computation. The default is 3 which is a correction factor for an ensemble of velocities with zero total linear momentum.

**Restrictions:** none

**Related commands:**

[temperature](#)

### Default:

The option defaults are `extra = 3`.

[LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#)

---

## 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 argument)

## 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`, `npt`, 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. For example,

```
variable t proc 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

---

[LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#)

---

## temperature command

### Syntax:

```
temperature ID group-ID style args keyword value ...
```

- ID = user-assigned name for the temperature
- group-ID = ID of group of atoms to compute the temperature for
- style = *full* or *partial* or *ramp*

```
full args = none
partial args = x y z
 x,y,z = 0 or 1
ramp args = vdim vlo vhi dim clo chi
 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 to the args
- keyword = *units*

```
units value = lattice or box
```

### Examples:

```
temperature mine peptide full
temperature new flow partial 1 1 0
temperature 2nd middle ramp vx 0 8 y 2 12 units lattice
```

### Description:

Define a method for computing the temperature of a group of atoms.

The ID of the temperature can be referred to in other commands which perform or modify temperature computations: thermo\_modify, velocity, fix\_modify, temp\_modify.

The style determines how the temperature is computed. The *full* style means  $KE = \text{dim}/2 N k T$ , 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.

The *partial* style uses the same formula as *full*, except entire dimensions can be eliminated from the kinetic energy computation. This can be useful for systems where atoms are flowing, and only the thermal temperature in the non-flow directions is desired. A "0" means do not use the component of velocity in that dimension when computing KE. In the example above with arguments 1 1 0, only x and y velocities (not z) would be used in computing KE and temperature.

The *ramp* style can be used to eliminate an imposed velocity on a system before computing thermal KE. The meaning of these arguments is the same as for the velocity command which was presumably used to impose the velocity.

The optional *units* keyword affects the coordinates (c1,c2) and velocities (vlo,vhi) defined for the *ramp* style. If *units* is set to *lattice* then distance and velocity are in lattice spacing units (e.g. lattice spacings per tau). If

*units* is *box* then the distance and velocity units are in Angstroms and sigma.

The *units* option affects the meaning of arguments for the *ramp* style. If *units* = *box*, the velocities and coordinates specified 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.

A temperature with ID = *default* is pre-defined by LAMMPS and uses to a *full* style computation on the *all* group of atoms. All operations in LAMMPS that compute temperatures use the *default* ID unless the input script changes it.

**Restrictions:** none

**Related commands:**

*thermo modify*, *velocity*, *fix modify*, *temp modify*

**Default:**

A temperature with ID = *default* is defined by LAMMPS, as if it had been specified as "temperature default all full".

LAMMPS WWW Site – LAMMPS Documentation – LAMMPS Commands

---

## thermo command

**Syntax:**

```
thermo N
```

- N = output thermodynamics every N timesteps

**Examples:**

```
thermo 100
```

**Description:**

Compute and print thermodynamics (temperature, energy, pressure) every N timesteps. A value of 0 will only compute thermodynamics at the beginning and end of a simulation.

**Restrictions:** none

**Related commands:**

*thermo style*, *thermo modify*

**Default:**

```
thermo 0
```

thermo command

## thermo\_modify command

### Syntax:

```
thermo_modify keyword value ...
```

- one or more keyword/value pairs may be listed
- keyword = *temp* or *lost* or *norm*

```
temp value = ID of temperature
lost value = error or warn or ignore
norm value = yes or no
```

### Examples:

```
thermo_modify temp mydef
thermo_modify lost no
```

### Description:

Set options for how thermodynamics are computed by LAMMPS. Different ways to compute temperature can be defined by the user (see the temperature command). The *temp* option allows you to specify which temperature computation will be used when thermodynamic info that uses temperature is computed and displayed (temperature, kinetic energy, pressure).

The *lost* option 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* option determines whether the thermodynamic print-out is normalized by the number of atoms or is the total summed across all atoms. Different atom styles have different defaults for this setting.

**Restrictions:** none

### Related commands:

[thermo](#), [thermo\\_style](#), [temperature](#)

### Default:

The option defaults are temp = default, lost = error, norm = no for unit style of *lj*, norm = yes for unit style of *real* and *metal*.

## thermo\_style command

### Syntax:

```
thermo_style style
```

- style = *one* or *multi* or *granular*

### Examples:

```
thermo_style multi
```

### Description:

Set the style in which thermodynamic data is printed to the screen and log file. Style *one* prints a one-line summary. All intra-molecular energy is included in E\_bond. Style *multi* prints 4 or 5 lines with a detailed energy breakdown. Style *granular* is used with [atom style granular](#).

### Restrictions: none

This command must come after the simulation box is defined by a [read\\_data](#), [read\\_restart](#), or [create\\_box](#) command.

The *granular* style 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.

### Related commands:

[thermo](#), [thermo\\_modify](#)

### Default:

```
thermo_style one
```

[LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#)

---

## 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.

When the run style is *respa*, dt is the timestep for the outer loop (largest) timestep.

**Restrictions:** none

**Related commands:**

run, run\_style respa

**Default:**

timestep = 0.005 tau for units = lj

timestep = 1.0 fmsec for units = real

timestep = 0.001 psec for units = metal

[LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#)

---

## 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

[LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#)

---

## unfix command

### Syntax:

```
unfix fix-ID
```

- fix-ID = ID of a previously defined fix

### Examples:

```
unfix 2
unfix lower-boundary
```

### Description:

Turn off a fix that was previously defined with a fix command.

**Restrictions:** none

### Related commands:

fix

**Default:** none

---

[LAMMPS WWW Site](#) – [LAMPS Documentation](#) – [LAMMPS Commands](#)

---

## units command

### Syntax:

```
units style
```

- style = *lj* or *real* or *metal*

### 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 style *lj*, all quantities are unitless:

- distance = sigma
- time = tau
- mass = one

- energy = epsilon
- velocity = sigma/tau
- force = epsilon/sigma
- temperature = reduced LJ temperature
- pressure = reduced LJ pressure
- charge = reduced LJ charge

For style *real*, these are the units:

- distance = Angstroms
- time = femtoseconds
- mass = grams/mole
- energy = Kcal/mole
- velocity = Angstroms/femtosecond
- force = Kcal/mole–Angstrom
- temperature = degrees K
- pressure = atmospheres
- charge = multiple of electron charge (+1.0 is a proton)

For style *metal*, these are the units:

- distance = Angstroms
- time = picoseconds
- mass = grams/mole
- energy = eV
- velocity = Angstroms/picosecond
- force = eV/Angstrom
- temperature = degrees K
- pressure = bars
- charge = multiple of electron charge (+1.0 is a proton)

This 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  $dt = 1.0 \text{ fmsec}$  and  $skin = 2.0 \text{ Angstroms}$ . For style *metal* these are  $dt = 0.001 \text{ psec}$  and  $skin = 2.0 \text{ Angstroms}$ .

### 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`

---

[LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#)

## variable command

### Syntax:

```
variable name style arg1 arg2 ...
```

- name = single lower-case character, 'a' thru 'z'
- style = *proc* or *index*
- arg1 thru argN = list of text strings

### Examples:

```
variable x proc 1 2 3 4
variable t proc 300.0 310.0 320.0 330.0
variable d index run1 run2 run3 run4 run5 run6 run7 run8
```

### Description:

This command sets values for a variable name that can be substituted for in later input script commands. As explained in [this section](#), all occurrences of \$X in an input script line are treated as a variable, where X is a single lower-case character from "a" to "z". Variables can also be set (with a single value) by using the command-line switch `-var`; see [this section](#) for details.

For *proc* style variables, the `-partition` command-line switch must be used when LAMMPS is run; see [this section](#). All processors in the Ith partition will substitute the Ith argument for the variable \$X. For example, in a simulation with 4 partitions and the variables *x* and *t* defined as above, these commands

```
read_data data.peptide.$x
temper 100000 100 $t settemp 0 45928
```

will run a parallel tempering simulation at 4 temperatures, using 4 different data files as inputs.

The substitution behavior of *index* style variables depends on whether LAMMPS is running on a single partition or multiple partitions. See [this section](#) for a discussion of the `-partition` command-line switch. See the [next](#) command for a discussion of how substitutions take place in both scenarios.

If a variable command is encountered when the variable has already been defined, the command is ignored. Thus allows an input script with a variable command to be processed multiple times; see the [jump](#) command. It also means that the use of the command-line switch `-var` will override a corresponding variable setting in the input script.

**Restrictions:** none

**Related commands:**

[next](#), [jump](#), [temper](#)

**Default:** none

---

[LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#)

## 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*

```
create args = temp seed
 temp = temperature value (temperature units)
 seed = random # seed (8 digits or less)
set args = vx vy vz
 vx,vy,vz = velocity value or NULL (velocity units)
scale args = 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 or more keyword/value pairs may be appended to the args
- 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 20 temp mytemp
```

### Description:

Set or change the velocities of a group of atoms. 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 temperature 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 20), 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 20 in this case), are assigned velocities equal to vlo or vhi (0.0 or 5.0 in this case).

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.

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. 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 user-defined temperature computation. If this option is not used, the default temperature (which has style *full*) is computed for the group of atoms specified in the velocity command. If the temperature should have degrees-of-freedom removed due to SHAKE constraints, then the appropriate fix shake command must be specified before the velocity command is issued.

The *loop* option is used by *create*. 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. It will produce the same set of velocities, independent of the number of processors, if atoms were read from a data file. It will not produce such independent velocities if atoms were created using the create atoms command. 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 will always produce different sets of velocities 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.

For all styles, no atoms are assigned z-component velocities if the simulation is 2d; see the dimension command.

**Restrictions:** none

velocity command

**Related commands:**

[fix shake](#), [lattice](#)

**Default:**

The option defaults are dist = uniform, sum = no, mom = yes, rot = no, temp = default, loop = all, and units = lattice.

[LAMMPS WWW Site](#) – [LAMMPS Documentation](#) – [LAMMPS Commands](#)

---

## **write\_restart command**

**Syntax:**

```
write_restart file
```

- file = name of file to write restart information to

**Examples:**

```
write_restart restart.equil
```

**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.

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.

**Restrictions:** none

**Related commands:**

[restart](#), [read\\_restart](#)

**Default:** none