LIGGGHTS(R)-WSU Documentation, Version 3.X

LIGGGHTS(R)-WSU DEM simulation engine

released by DCS Computing Gmbh, Linz, Austria, www.dcs-computing.com , office@dcs-computing.com

LIGGGHTS(R)-WSU is open-source, distributed under the terms of the GNU Public License, version 2 or later. LIGGGHTS(R)-WSU is part of CFDEM(R)project: www.liggghts.com | www.cfdem.com

Core developer and main author: Christoph Kloss, christoph.kloss@dcs-computing.com

LIGGGHTS(R)-WSU is an Open Source Discrete Element Method Particle Simulation Software, distributed by DCS Computing GmbH, Linz, Austria. LIGGGHTS (R) and CFDEM(R) are registered trade marks of DCS Computing GmbH, the producer of the LIGGGHTS (R) software and the CFDEM(R)coupling software See http://www.cfdem.com/terms-trademark-policy for details.

LIGGGHTS (R) Version info:

All LIGGGHTS (R) versions are based on a specific version of LIGGGHTS (R), as printed in the file src/version.h LIGGGHTS (R) versions are identidied by a version number (e.g. '3.0'), a branch name (which is 'LIGGGHTS(R)-WSU' for your release of LIGGGHTS), compilation info (date / time stamp and user name), and a LAMMPS version number (which is the LAMMPS version that the LIGGGHTS(R)-WSU release is based on). The LAMMPS "version" is the date when it was released, such as 1 May 2010.

If you browse the HTML doc pages on the LIGGGHTS(R)-WSU WWW site, they always describe the most current version of LIGGGHTS(R)-WSU. If you browse the HTML doc pages included in your tarball, they describe the version you have.

LIGGGHTS (R) and its ancestor LAMMPS:

LAMMPS is a classical molecular dynamics simulation code designed to run efficiently on parallel computers. It was developed at Sandia National Laboratories, a US Department of Energy facility, with funding from the DOE. It is an open-source code, distributed freely under the terms of the GNU Public License (GPL). The primary developers of LAMMPS are Steve Plimpton, Aidan Thompson, and Paul Crozier. The LAMMPS WWW Site at http://lammps.sandia.gov has more information about LAMMPS.

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

Once you are familiar with LIGGGHTS(R)-WSU, you may want to bookmark this page since it gives quick access to documentation for all LIGGGHTS(R)-WSU commands.

atom_modify command

Syntax:

```
atom_modify keyword values ...
```

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

```
  map value = array or hash
    first value = group-ID = group whose atoms will appear first in internal atom lists
    sort values = Nfreq binsize
      Nfreq = sort atoms spatially every this many time steps
      binsize = bin size for spatial sorting (distance units)
```

Examples:

```
atom_modify map hash
atom_modify map array sort 10000 2.0
atom_modify first colloid
```

Description:

Modify properties of the atom style selected within LIGGGHTS(R)-WSU.

The *map* keyword determines how atom ID lookup is done for molecular problems. Lookups are performed by bond (angle, etc) routines in LIGGGHTS(R)-WSU to find the local atom index associated with a global atom ID. When the *array* value is used, each processor stores a lookup table of length N, where N is the total # of atoms in the system. This is the fastest method for most simulations, but a processor can run out of memory to store the table for very large simulations. The *hash* value uses a hash table to perform the lookups. This method can be slightly slower than the *array* method, but its memory cost is proportional to N/P on each processor, where P is the total number of processors running the simulation.

The *first* keyword allows a group to be specified whose atoms will be maintained as the first atoms in each processor's list of owned atoms. This in only useful when the specified group is a small fraction of all the atoms, and there are other operations LIGGGHTS(R)-WSU is performing that will be sped-up significantly by being able to loop over the smaller set of atoms. Otherwise the reordering required by this option will be a net slow-down. The neigh_modify include and communicate group commands are two examples of commands that require this setting to work efficiently. Several fixes, most notably time integration fixes like fix nve, also take advantage of this setting if the group they operate on is the group specified by this command. Note that specifying "all" as the group-ID effectively turns off the *first* option.

It is OK to use the *first* keyword with a group that has not yet been defined, e.g. to use the atom_modify first command at the beginning of your input script. LIGGGHTS(R)-WSU does not use the group until a simullation is run.

The *sort* keyword turns on a spatial sorting or reordering of atoms within each processor's sub-domain every *Nfreq* timesteps. If *Nfreq* is set to 0, then sorting is turned off. Sorting can improve cache performance and thus speed-up a LIGGGHTS(R)-WSU simulation, as discussed in a paper by (Meloni). Its efficacy depends on the problem size (atoms/processor), how quickly the system becomes disordered, and various other factors. As a general rule, sorting is typically more effective at speeding up simulations of liquids as opposed to solids. In tests we have done, the speed-up can range from zero to 3-4x.

Reordering is peformed every *Nfreq* timesteps during a dynamics run or iterations during a minimization. More precisely, reordering occurs at the first reneighboring that occurs after the target timestep. The reordering is performed locally by each processor, using bins of the specified *binsize*. If *binsize* is set to 0.0, then a binsize equal to half the [neighbor](#) cutoff distance (force cutoff plus skin distance) is used, which is a reasonable value. After the atoms have been binned, they are reordered so that atoms in the same bin are adjacent to each other in the processor's 1d list of atoms.

The goal of this procedure is for atoms to put atoms close to each other in the processor's one-dimensional list of atoms that are also near to each other spatially. This can improve cache performance when pairwise interactions and neighbor lists are computed. Note that if bins are too small, there will be few atoms/bin. Likewise if bins are too large, there will be many atoms/bin. In both cases, the goal of cache locality will be undermined.

IMPORTANT NOTE: Running a simulation with sorting on versus off should not change the simulation results in a statistical sense. However, a different ordering will induce round-off differences, which will lead to diverging trajectories over time when comparing two simulations. Various commands, particularly those which use random numbers, may generate (statistically identical) results which depend on the order in which atoms are processed. The order of atoms in a [dump](#) file will also typically change if sorting is enabled.

Restrictions:

The map keyword can only be used before the simulation box is defined by a [read_data](#) or [create_box](#) command.

The *first* and *sort* options cannot be used together. Since sorting is on by default, it will be turned off if the *first* keyword is used with a group-ID that is not "all".

Related commands: none

Default:

By default, non-molecular problems do not allocate maps. For molecular problems, the option default is map = array. By default, a "first" group is not defined. By default, sorting is enabled with a frequency of 1000 and a binsize of 0.0, which means the neighbor cutoff will be used to set the bin size.

(Meloni) Meloni, Rosati and Colombo, J Chem Phys, 126, 121102 (2007).

atom_style command

Syntax:

```
atom_style style args
```

- style = *bond* or *charge* or *ellipsoid* or *full* or *line* or *molecular* or *sphere* or *granular* or *bond/gran* or *tri* or *hybrid* or *sph*

```
  args = none for any style except body and hybrid
  body args = bstyle bstyle-args
    bstyle = style of body particles
    bstyle-args = additional arguments specific to the bstyle
                  see the body doc page for details
  hybrid args = list of one or more sub-styles, each with their args
```

Examples:

```
atom_style bond
atom_style sphere
atom_style superquadric (not available yet in the PUBLIC version)
atom_style hybrid sphere bond
```

Description:

Define what style of atoms to use in a simulation. This determines what attributes are associated with the atoms. This command must be used before a simulation is setup via a [read_data](#), [read_restart](#), or [create_box](#) command.

Once a style is assigned, it cannot be changed, so use a style general enough to encompass all attributes. E.g. with style *bond*, angular terms cannot be used or added later to the model. It is OK to use a style more general than needed, though it may be slightly inefficient.

The choice of style affects what quantities are stored by each atom, what quantities are communicated between processors to enable forces to be computed, and what quantities are listed in the data file read by the [read_data](#) command.

These are the additional attributes of each style and the typical kinds of physical systems they are used to model. All styles store coordinates, velocities, atom IDs and types. See the [read_data](#), [create_atoms](#), and [set](#) commands for info on how to set these various quantities.

bond	bonds	bead-spring polymers
bond/gran	number of bonds and bond information	granular bond models
charge	charge	atomic system with charges
ellipsoid	shape, quaternion, angular momentum	aspherical particles
line	end points, angular velocity	rigid bodies
sph	q(pressure), density	SPH particles
molecular	bonds, angles, dihedrals, impropers	

		uncharged molecules
sphere or granular	diameter, mass, angular velocity	granular models
superquadric	semi-axes, roundness/blockiness parameters, mass, angular velocity, quaternion	granular models
tri	corner points, angular momentum	rigid bodies AWPMD

IMPORTANT NOTE: It is possible to add some attributes, such as a molecule ID, to atom styles that do not have them via the fix property/atom command. This command also allows new custom attributes consisting of extra integer or floating-point values to be added to atoms. See the fix property/atom doc page for examples of cases where this is useful and details on how to initialize, access, and output the custom values.

All of the styles assign mass to particles on a per-type basis, using the mass command, except for *sphere or granular* styles. They assign mass to individual particles on a per-particle basis.

For the *sphere* style, the particles are spheres and each stores a per-particle diameter and mass. If the diameter > 0.0, the particle is a finite-size sphere. If the diameter = 0.0, it is a point particle. This is typically used for granular models. Instead of *sphere*, keyword *granular* can be used.

For the *bond/gran* style, the number of granular bonds per atom is stored, and the information associated to it: the type of each bond, the ID of the bonded partner atom and the so-called bond history. The bond history is similar to the contact history for granular interaction, it stores the internal state of the bond. What exactly is stored in this internal state is defined by the granular bond style used. There are 2 parameters: The number of bond types, and the maximum number of bonds that each atom can have. For each bond type, the parameters have to be specified via the bond_coeff command (see example here) Note that *bond/gran* is an experimental code which is may not be available in your release of LIGGGHTS. An example for the sytnax is given below:

```
atom_style bond/gran n_bondtypes 1 bonds_per_atom 6
```

For the *ellipsoid* style, the particles are ellipsoids and each stores a flag which indicates whether it is a finite-size ellipsoid or a point particle. If it is an ellipsoid, it also stores a shape vector with the 3 diamters of the ellipsoid and a quaternion 4-vector with its orientation.

For the *line* style, the particles are idealized line segments and each stores a per-particle mass and length and orientation (i.e. the end points of the line segment).

For the *tri* style, the particles are planar triangles and each stores a per-particle mass and size and orientation (i.e. the corner points of the triangle).

Typically, simulations require only a single (non-hybrid) atom style. If some atoms in the simulation do not have all the properties defined by a particular style, use the simplest style that defines all the needed properties by any atom. For example, if some atoms in a simulation are charged, but others are not, use the *charge* style. If some atoms have bonds, but others do not, use the *bond* style.

The only scenario where the *hybrid* style is needed is if there is no single style which defines all needed properties of all atoms. For example, if you want dipolar particles which will rotate due to torque, you would need to use "atom_style hybrid sphere dipole". When a hybrid style is used, atoms store and communicate the union of all quantities implied by the individual styles.

LIGGGHTS(R)-WSU can be extended with new atom styles as well as new body 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 *superquadric* style is in preparation and not yet available in the PUBLIC version

The *bond*, *molecular* styles are part of the MOLECULAR package. The *line* and *tri* styles are part of the ASPHERE package. They are only enabled if LIGGGHTS(R)-WSU was built with that package. See the [Making LIGGGHTS(R)-WSU](#) section for more info.

Related commands:

[read_data](#), [pair_style](#)

Default: none

bond_coeff command

Syntax:

```
bond_coeff N args
```

- N = bond type (see asterisk form below)
- args = coefficients for one or more bond types

Examples:

```
bond_coeff 5 80.0 1.2
bond_coeff * 30.0 1.5 1.0 1.0
bond_coeff 1*4 30.0 1.5 1.0 1.0
bond_coeff 1 harmonic 200.0 1.0
```

Description:

Specify the bond force field coefficients for one or more bond types. The number and meaning of the coefficients depends on the bond style. Bond coefficients can also be set in the data file read by the [read_data](#) command or in a restart file.

N can be specified in one of two ways. An explicit numeric value can be used, as in the 1st example above. Or a wild-card asterisk can be used to set the coefficients for multiple bond types. This takes the form "*" or "*n" or "n*" or "m*n". If N = the number of bond types, then an asterisk with no numeric values means all types from 1 to N. A leading asterisk means all types from 1 to n (inclusive). A trailing asterisk means all types from n to N (inclusive). A middle asterisk means all types from m to n (inclusive).

Note that using a bond_coeff command can override a previous setting for the same bond type. For example, these commands set the coeffs for all bond types, then overwrite the coeffs for just bond type 2:

```
bond_coeff * 100.0 1.2
bond_coeff 2 200.0 1.2
```

A line in a data file that specifies bond coefficients uses the exact same format as the arguments of the bond_coeff command in an input script, except that wild-card asterisks should not be used since coefficients for all N types must be listed in the file. For example, under the "Bond Coeffs" section of a data file, the line that corresponds to the 1st example above would be listed as

```
5 80.0 1.2
```

Here is an alphabetic list of bond styles defined in LIGGGHTS(R)-WSU. Click on the style to display the formula it computes and coefficients specified by the associated [bond_coeff](#) command.

Note that here are also additional bond styles submitted by users which are included in the LIGGGHTS(R)-WSU distribution. The list of these with links to the individual styles are given in the bond section of [this page](#).

- [bond_style none](#) - turn off bonded interactions
- [bond_style hybrid](#) - define multiple styles of bond interactions

- [bond_style harmonic](#) - harmonic bond

Restrictions:

This command must come after the simulation box is defined by a read_data, read_restart, or create_box command.

A bond style must be defined before any bond coefficients are set, either in the input script or in a data file.

Related commands:

bond_style

Default: none

bond_style harmonic command

Syntax:

```
bond_style harmonic
```

Examples:

```
bond_style harmonic
bond_coeff 5 80.0 1.2
```

Description:

The *harmonic* bond style uses the potential

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

where r0 is the equilibrium bond distance. Note that the usual 1/2 factor is included in K.

The following coefficients must be defined for each bond type via the bond_coeff command as in the example above, or in the data file or restart files read by the read_data or read_restart commands:

- K (energy/distance^2)
- r0 (distance)

Restrictions:

This bond style can only be used if LIGGGHTS(R)-WSU was built with the MOLECULAR package (which it is by default). See the Making LIGGGHTS(R)-WSU section for more info on packages.

Related commands:

bond_coeff, delete_bonds

Default: none

bond_style hybrid command

Syntax:

```
bond_style hybrid style1 style2 ...
```

- style1,style2 = list of one or more bond styles

Examples:

```
bond_style hybrid harmonic fene
bond_coeff 1 harmonic 80.0 1.2
bond_coeff 2* fene 30.0 1.5 1.0 1.0
```

Description:

The *hybrid* style enables the use of multiple bond styles in one simulation. A bond style is assigned to each bond type. For example, bonds in a polymer flow (of bond type 1) could be computed with a *fene* potential and bonds in the wall boundary (of bond type 2) could be computed with a *harmonic* potential. The assignment of bond type to style is made via the bond_coeff command or in the data file.

In the bond_coeff commands, the name of a bond style must be added after the bond type, with the remaining coefficients being those appropriate to that style. In the example above, the 2 bond_coeff commands set bonds of bond type 1 to be computed with a *harmonic* potential with coefficients 80.0, 1.2 for K, r0. All other bond types (2-N) are computed with a *fene* potential with coefficients 30.0, 1.5, 1.0, 1.0 for K, R0, epsilon, sigma.

If bond coefficients are specified in the data file read via the read_data command, then the same rule applies. E.g. "harmonic" or "fene" must be added after the bond type, for each line in the "Bond Coeffs" section, e.g.

```
Bond Coeffs

1 harmonic 80.0 1.2
2 fene 30.0 1.5 1.0 1.0
...
```

A bond style of *none* with no additional coefficients can be used in place of a bond style, either in a input script bond_coeff command or in the data file, if you desire to turn off interactions for specific bond types.

Restrictions:

This bond style can only be used if LIGGGHTS(R)-WSU was built with the MOLECULAR package (which it is by default). See the Making LIGGGHTS(R)-WSU section for more info on packages.

Unlike other bond styles, the hybrid bond style does not store bond coefficient info for individual sub-styles in a binary restart files. Thus when retarting a simulation from a restart file, you need to re-specify bond_coeff commands.

Related commands:

bond_coeff, delete_bonds

Default: none

bond_style none command

Syntax:

```
bond_style none
```

Examples:

```
bond_style none
```

Description:

Using a bond style of none means bond forces are not computed, even if pairs of bonded atoms were listed in the data file read by the read_data command.

Restrictions: none

Related commands: none

Default: none

bond_style command

Syntax:

```
bond_style style args
```

- style = *none* or *hybrid* or *class2* or *fene* or *fene/expand* or *harmonic* or *morse* or *nonlinear* or *quartic*

```
args = none for any style except hybrid
hybrid args = list of one or more styles
```

Examples:

```
bond_style harmonic
bond_style fene
bond_style hybrid harmonic fene
```

Description:

Set the formula(s) LIGGGHTS(R)-WSU uses to compute bond interactions between pairs of atoms. In LIGGGHTS(R)-WSU, a bond differs from a pairwise interaction, which are set via the [pair_style](#) command. Bonds are defined between specified pairs of atoms and remain in force for the duration of the simulation (unless the bond breaks which is possible in some bond potentials). The list of bonded atoms is read in by a [read_data](#) or [read_restart](#) command from a data or restart file. By contrast, pair potentials are typically defined between all pairs of atoms within a cutoff distance and the set of active interactions changes over time.

Hybrid models where bonds are computed using different bond potentials can be setup using the *hybrid* bond style.

The coefficients associated with a bond style can be specified in a data or restart file or via the [bond_coeff](#) command.

All bond potentials store their coefficient data in binary restart files which means bond_style and [bond_coeff](#) commands do not need to be re-specified in an input script that restarts a simulation. See the [read_restart](#) command for details on how to do this. The one exception is that bond_style *hybrid* only stores the list of sub-styles in the restart file; bond coefficients need to be re-specified.

IMPORTANT NOTE: When both a bond and pair style is defined, the [special_bonds](#) command often needs to be used to turn off (or weight) the pairwise interaction that would otherwise exist between 2 bonded atoms.

In the formulas listed for each bond style, *r* is the distance between the 2 atoms in the bond.

Here is an alphabetic list of bond styles defined in LIGGGHTS(R)-WSU. Click on the style to display the formula it computes and coefficients specified by the associated [bond_coeff](#) command.

Note that there are also additional bond styles submitted by users which are included in the LIGGGHTS(R)-WSU distribution. The list of these with links to the individual styles are given in the bond section of [this page](#).

- [bond_style none](#) - turn off bonded interactions
- [bond_style hybrid](#) - define multiple styles of bond interactions

- [bond_style harmonic](#) - harmonic bond

Restrictions:

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

Most bond styles are part of the MOLECULAR package. They are only enabled if LIGGGHTS(R)-WSU was built with that package. See the [Making LIGGGHTS(R)-WSU](#) section for more info on packages. The doc pages for individual bond potentials tell if it is part of a package.

Related commands:

[bond_coeff](#), [delete_bonds](#)

Default:

bond_style none

boundary command

Syntax:

```
boundary x y z
```

- x,y,z = *p* or *s* or *f* or *m*, one or two letters

```
p is periodic
  f is non-periodic and fixed
  s is non-periodic and shrink-wrapped
  m is non-periodic and shrink-wrapped with a minimum value
```

Examples:

```
boundary p p f
boundary p fs p
boundary s f fm
```

Description:

Set the style of boundaries for the global simulation box in each dimension. A single letter assigns the same style to both the lower and upper face of the box. Two letters assigns the first style to the lower face and the second style to the upper face. The initial size of the simulation box is set by the [read_data](#), [read_restart](#), or [create_box](#) commands.

The style *p* means the box is periodic, so that particles interact across the boundary, and they can exit one end of the box and re-enter the other end. A periodic dimension can change in size due to constant pressure boundary conditions or box deformation (see the [fix npt](#) and [fix deform](#) commands). The *p* style must be applied to both faces of a dimension.

The styles *f*, *s*, and *m* mean the box is non-periodic, so that particles do not interact across the boundary and do not move from one side of the box to the other. For style *f*, the position of the face is fixed. If an atom moves outside the face it may be lost. For style *s*, the position of the face is set so as to encompass the atoms in that dimension (shrink-wrapping), no matter how far they move. For style *m*, shrink-wrapping occurs, but is bounded by the value specified in the data or restart file or set by the [create_box](#) command. For example, if the upper z face has a value of 50.0 in the data file, the face will always be positioned at 50.0 or above, even if the maximum z-extent of all the atoms becomes less than 50.0.

For triclinic (non-orthogonal) simulation boxes, if the 2nd dimension of a tilt factor (e.g. y for xy) is periodic, then the periodicity is enforced with the tilt factor offset. If the 1st dimension is shrink-wrapped, then the shrink wrapping is applied to the tilted box face, to encompass the atoms. E.g. for a positive xy tilt, the xlo and xhi faces of the box are planes tilting in the +y direction as y increases. These tilted planes are shrink-wrapped around the atoms to determine the x extent of the box.

See [Section_howto 12](#) of the doc pages for a geometric description of triclinic boxes, as defined by LIGGGHTS(R)-WSU, and how to transform these parameters to and from other commonly used triclinic representations.

IMPORTANT NOTE: If mesh walls (e.g. [fix mesh/surface](#)) are used, not only atom positions, but also the mesh nodes are used for setting the boundaries.

Restrictions:

This command cannot be used after the simulation box is defined by a read_data or create_box command or read_restart command. See the change_box command for how to change the simulation box boundaries after it has been defined.

For 2d simulations, the z dimension must be periodic.

Related commands:

See the thermo_modify command for a discussion of lost atoms.

Default:

```
boundary p p p
```

box command

Syntax:

```
box keyword value ...
```

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

    tilt value = *small* or *large*

Examples:

```
box tilt large
box tilt small
```

Description:

Set attributes of the simulation box.

For triclinic (non-orthogonal) simulation boxes, the *tilt* keyword allows simulation domains to be created with arbitrary tilt factors, e.g. via the [create_box](#) or [read_data](#) commands. Tilt factors determine how skewed the triclinic box is; see [this section](#) of the manual for a discussion of triclinic boxes in LIGGGHTS(R)-WSU.

LIGGGHTS(R)-WSU normally requires that no tilt factor can skew the box more than half the distance of the parallel box length, which is the 1st dimension in the tilt factor (x for xz). If *tilt* is set to *small*, which is the default, then an error will be generated if a box is created which exceeds this limit. If *tilt* is set to *large*, then no limit is enforced. You can create a box with any tilt factors you wish.

Note that if a simulation box has a large tilt factor, LIGGGHTS(R)-WSU will run less efficiently, due to the large volume of communication needed to acquire ghost atoms around a processor's irregular-shaped sub-domain. For extreme values of tilt, LIGGGHTS(R)-WSU may also lose atoms and generate an error.

Restrictions:

This command cannot be used after the simulation box is defined by a [read_data](#) or [create_box](#) command or [read_restart](#) command.

Related commands: none

Default:

The default value is tilt = small.

change_box command

Syntax:

```
change_box group-ID parameter args ... keyword args ...
```

- group-ID = ID of group of atoms to (optionally) displace
- one or more parameter/arg pairs may be appended

```
  parameter = x or y or z or xy or xz or yz or boundary or ortho or triclinic or set or remap
    x, y, z args = style value(s)
      style = final or delta or scale or volume
        final values = lo hi
          lo hi = box boundaries after displacement (distance units)
        delta values = dlo dhi
          dlo dhi = change in box boundaries after displacement (distance units)
        scale values = factor
          factor = multiplicative factor for change in box length after displacement
        volume value = none = adjust this dim to preserve volume of system
    xy, xz, yz args = style value
      style = final or delta
        final value = tilt
          tilt = tilt factor after displacement (distance units)
        delta value = dtilt
          dtilt = change in tilt factor after displacement (distance units)
    boundary args = 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
    ortho args = none = change box to orthogonal
    triclinic args = none = change box to triclinic
    set args = none = store state of current box
    remap args = none = remap atom coords from last saved state to current box
```

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

```
  units value = lattice or box
      lattice = distances are defined in lattice units
      box = distances are defined in simulation box units
```

Examples:

```
change_box all xy final -2.0 z final 0.0 5.0 boundary p p f remap units box
change_box all x scale 1.1 y volume z volume remap
```

Description:

Change the volume and/or shape and/or boundary conditions for the simulation box. Orthogonal simulation boxes have 3 adjustable size parameters (x,y,z). Triclinic (non-orthogonal) simulation boxes have 6 adjustable size/shape parameters (x,y,z,xy,xz,yz). Any or all of them can be adjusted independently by this command. Thus it can be used to expand or contract a box, or to apply a shear strain to a non-orthogonal box. It can also be used to change the boundary conditions for the simulation box, similar to the [boundary](#) command.

The size and shape of the initial simulation box are specified by the [create_box](#) or [read_data](#) or [read_restart](#) command used to setup the simulation. The size and shape may be altered by subsequent runs, e.g. by use of

the [fix npt](#) or [fix deform](#) commands. The [create_box](#), [read_data](#), and [read_restart](#) commands also determine whether the simulation box is orthogonal or triclinic and their doc pages explain the meaning of the xy,xz,yz tilt factors.

See [Section_howto 12](#) of the doc pages for a geometric description of triclinic boxes, as defined by LIGGGHTS(R)-WSU, and how to transform these parameters to and from other commonly used triclinic representations.

The keywords used in this command are applied sequentially to the simulation box and the atoms in it, in the order specified.

Before the sequence of keywords are invoked, the current box size/shape is stored, in case a *remap* keyword is used to map the atom coordinates from a previously stored box size/shape to the current one.

After all the keywords have been processed, any shrink-wrap boundary conditions are invoked (see the [boundary](#) command) which may change simulation box boundaries, and atoms are migrated to new owning processors.

IMPORTANT NOTE: Unlike the earlier "displace_box" version of this command, atom remapping is NOT performed by default. This command allows remapping to be done in a more general way, exactly when you specify it (zero or more times) in the sequence of transformations. Thus if you do not use the *remap* keyword, atom coordinates will not be changed even if the box size/shape changes. If a uniformly strained state is desired, the *remap* keyword should be specified.

IMPORTANT NOTE: It is possible to lose atoms with this command. E.g. by changing the box without remapping the atoms, and having atoms end up outside of non-periodic boundaries. It is also possible to alter bonds between atoms straddling a boundary in bad ways. E.g. by converting a boundary from periodic to non-periodic. It is also possible when remapping atoms to put them (nearly) on top of each other. E.g. by converting a boundary from non-periodic to periodic. All of these will typically lead to bad dynamics and/or generate error messages.

IMPORTANT NOTE: The simulation box size/shape can be changed by arbitrarily large amounts by this command. This is not a problem, except that the mapping of processors to the simulation box is not changed from its initial 3d configuration; see the [processors](#) command. Thus, if the box size/shape changes dramatically, the mapping of processors to the simulation box may not end up as optimal as the initial mapping attempted to be.

IMPORTANT NOTE: Because the keywords used in this command are applied one at a time to the simulation box and the atoms in it, care must be taken with triclinic cells to avoid exceeding the limits on skew after each transformation in the sequence. If skew is exceeded before the final transformation this can be avoided by changing the order of the sequence, or breaking the transformation into two or more smaller transformations. For more information on the allowed limits for box skew see the discussion on triclinic boxes on [this page](#).

For the *x*, *y*, and *z* parameters, this is the meaning of their styles and values.

For style *final*, the final lo and hi box boundaries of a dimension are specified. The values can be in lattice or box distance units. See the discussion of the units keyword below.

For style *delta*, plus or minus changes in the lo/hi box boundaries of a dimension are specified. The values can be in lattice or box distance units. See the discussion of the units keyword below.

For style *scale*, a multiplicative factor to apply to the box length of a dimension is specified. For example, if the initial box length is 10, and the factor is 1.1, then the final box length will be 11. A factor less than 1.0 means compression.

change_box command   17

The *volume* style changes the specified dimension in such a way that the overall box volume remains constant with respect to the operation performed by the preceding keyword. The *volume* style can only be used following a keyword that changed the volume, which is any of the *x*, *y*, *z* keywords. If the preceding keyword "key" had a *volume* style, then both it and the current keyword apply to the keyword preceding "key". I.e. this sequence of keywords is allowed:

```
change_box all x scale 1.1 y volume z volume
```

The *volume* style changes the associated dimension so that the overall box volume is unchanged relative to its value before the preceding keyword was invoked.

If the following command is used, then the z box length will shrink by the same 1.1 factor the x box length was increased by:

```
change_box all x scale 1.1 z volume
```

If the following command is used, then the y,z box lengths will each shrink by sqrt(1.1) to keep the volume constant. In this case, the y,z box lengths shrink so as to keep their relative aspect ratio constant:

```
change_box all"x scale 1.1 y volume z volume
```

If the following command is used, then the final box will be a factor of 10% larger in x and y, and a factor of 21% smaller in z, so as to keep the volume constant:

```
change_box all x scale 1.1 z volume y scale 1.1 z volume
```

IMPORTANT NOTE: For solids or liquids, when one dimension of the box is expanded, it may be physically undesirable to hold the other 2 box lengths constant since that implies a density change. For solids, adjusting the other dimensions via the *volume* style may make physical sense (just as for a liquid), but may not be correct for materials and potentials whose Poisson ratio is not 0.5.

For the *scale* and *volume* styles, the box length is expanded or compressed around its mid point.

For the *xy*, *xz*, and *yz* parameters, this is the meaning of their styles and values. Note that changing the tilt factors of a triclinic box does not change its volume.

For style *final*, the final tilt factor is specified. The value can be in lattice or box distance units. See the discussion of the units keyword below.

For style *delta*, a plus or minus change in the tilt factor is specified. The value can be in lattice or box distance units. See the discussion of the units keyword below.

All of these styles change the xy, xz, yz tilt factors. In LIGGGHTS(R)-WSU, tilt factors (xy,xz,yz) for triclinic boxes are required to be no more than half the distance of the parallel box length. For example, if xlo = 2 and xhi = 12, then the x box length is 10 and the xy tilt factor must be between -5 and 5. Similarly, both xz and yz must be between -(xhi-xlo)/2 and +(yhi-ylo)/2. Note that this is not a limitation, since if the maximum tilt factor is 5 (as in this example), then configurations with tilt = ..., -15, -5, 5, 15, 25, ... are all equivalent. Any tilt factor specified by this command must be within these limits.

The *boundary* keyword takes arguments that have exactly the same meaning as they do for the [boundary] command. 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 style *p* means the box is periodic; the other styles mean non-periodic. For style *f*, the position of the face is fixed. 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 current box edge in that dimension, so that the box will become no smaller. See the boundary command for more explanation of these style options.

Note that the "boundary" command itself can only be used before the simulation box is defined via a read_data or create_box or read_restart command. This command allows the boundary conditions to be changed later in your input script. Also note that the read_restart will change boundary conditions to match what is stored in the restart file. So if you wish to change them, you should use the change_box command after the read_restart command.

The *ortho* and *triclinic* keywords convert the simulation box to be orthogonal or triclinic (non-orthongonal). See this section for a discussion of how non-orthongal boxes are represented in LIGGGHTS(R)-WSU.

The simulation box is defined as either orthogonal or triclinic when it is created via the create_box, read_data, or read_restart commands.

These keywords allow you to toggle the existing simulation box from orthogonal to triclinic and vice versa. For example, an initial equilibration simulation can be run in an orthogonal box, the box can be toggled to triclinic.

If the simulation box is currently triclinic and has non-zero tilt in xy, yz, or xz, then it cannot be converted to an orthogonal box.

The *set* keyword saves the current box size/shape. This can be useful if you wish to use the *remap* keyword more than once or if you wish it to be applied to an intermediate box size/shape in a sequence of keyword operations. Note that the box size/shape is saved before any of the keywords are processed, i.e. the box size/shape at the time the create_box command is encountered in the input script.

The *remap* keyword remaps atom coordinates from the last saved box size/shape to the current box state. For example, if you stretch the box in the x dimension or tilt it in the xy plane via the *x* and *xy* keywords, then the *remap* commmand will dilate or tilt the atoms to conform to the new box size/shape, as if the atoms moved with the box as it deformed.

Note that this operation is performed without regard to periodic boundaries. Also, any shrink-wrapping of non-periodic boundaries (see the boundary command) occurs after all keywords, including this one, have been processed.

Only atoms in the specified group are remapped.

The *units* keyword determines the meaning of the distance units used to define various arguments. A *box* value selects standard distance units as defined by the units command, e.g. Angstroms for units = real or metal. A *lattice* value means the distance units are in lattice spacings. The lattice command must have been previously used to define the lattice spacing.

Restrictions:

If you use the *ortho* or *triclinic* keywords, then at the point in the input script when this command is issued, no dumps can be active, nor can a fix ave/spatial or fix deform be active. This is because these commands test whether the simulation box is orthogonal when they are first issued. Note that these commands can be used in your script before a change_box command is issued, so long as an undump or unfix command is also used to turn them off.

Related commands:

[fix deform](), [boundary]()

Default:

The option default is units = box.

[fix deform](), [boundary]()

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 LIGGGHTS(R)-WSU. Once a clear command has been executed, it is as if LIGGGHTS(R)-WSU were starting over, with only the exceptions noted below. This command enables multiple jobs to be run sequentially from one input script.

These settings are not affected by a clear command: the working directory (shell command), log file status (log command), echo status (echo command), and input script variables (variable command).

Restrictions: none

Related commands: none

Default: none

communicate command

Syntax:

```
communicate style keyword value ...
```

- style = *single* or *multi*
- zero or more keyword/value pairs may be appended
- keyword = *cutoff* or *group* or *vel*

```
  cutoff value = Rcut (distance units) = communicate atoms from this far away
    group value = group-ID = only communicate atoms in the group
    vel value = yes or no = do or do not communicate velocity info with ghost atoms
```

Examples:

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

Description:

This command sets the style of inter-processor communication that occurs each timestep as atom coordinates and other properties are exchanged between neighboring processors and stored as properties of ghost atoms.

The default style is *single* which means each processor acquires information for ghost atoms that are within a single distance from its sub-domain. The distance is the maximum of the neighbor cutoff for all atom type pairs.

For many systems this is an efficient algorithm, but for systems with widely varying cutoffs for different type pairs, the *multi* style can be faster. In this case, each atom type is assigned its own distance cutoff for communication purposes, and fewer atoms will be communicated. However, for granular systems optimization is automatically performed with the *single* style, so *multi* is not necessary/available for granular systems. See the [neighbor multi](#) command for a neighbor list construction option that may also be beneficial for simulations of this kind.

The *cutoff* option allows you to set a ghost cutoff distance, which is the distance from the borders of a processor's sub-domain at which ghost atoms are acquired from other processors. By default the ghost cutoff = neighbor cutoff = pairwise force cutoff + neighbor skin. See the [neighbor](#) command for more information about the skin distance. If the specified Rcut is greater than the neighbor cutoff, then extra ghost atoms will be acquired. If it is smaller, the ghost cutoff is set to the neighbor cutoff.

These are simulation scenarios in which it may be useful or even necessary to set a ghost cutoff > neighbor cutoff:

- a single polymer chain with bond interactions, but no pairwise interactions
- bonded interactions (e.g. dihedrals) extend further than the pairwise cutoff
- ghost atoms beyond the pairwise cutoff are needed for some computation

In the first scenario, a pairwise potential is not defined. Thus the pairwise neighbor cutoff will be 0.0. But ghost atoms are still needed for computing bond, angle, etc interactions between atoms on different processors, or when the interaction straddles a periodic boundary.

The appropriate ghost cutoff depends on the newton bond setting. For newton bond *off*, the distance needs to be the furthest distance between any two atoms in the bond. E.g. the distance between 1-4 atoms in a dihedral. For newton bond *on*, the distance between the central atom in the bond, angle, etc and any other atom is sufficient. E.g. the distance between 2-4 atoms in a dihedral.

In the second scenario, a pairwise potential is defined, but its neighbor cutoff is not sufficiently long enough to enable bond, angle, etc terms to be computed. As in the previous scenario, an appropriate ghost cutoff should be set.

In the last scenario, a fix or compute or pairwise potential needs to calculate with ghost atoms beyond the normal pairwise cutoff for some computation it performs (e.g. locate neighbors of ghost atoms in a multibody pair potential). Setting the ghost cutoff appropriately can insure it will find the needed atoms.

IMPORTANT NOTE: In these scenarios, if you do not set the ghost cutoff long enough, and if there is only one processor in a periodic dimension (e.g. you are running in serial), then LIGGGHTS(R)-WSU may "find" the atom it is looking for (e.g. the partner atom in a bond), that is on the far side of the simulation box, across a periodic boundary. This will typically lead to bad dynamics (i.e. the bond length is now the simulation box length). To detect if this is happening, see the neigh_modify cluster command.

The *group* option will limit communication to atoms in the specified group. This can be useful for models where no ghost atoms are needed for some kinds of particles. All atoms (not just those in the specified group) will still migrate to new processors as they move. The group specified with this option must also be specified via the atom_modify first command.

The *vel* option enables velocity information to be communicated with ghost particles. Depending on the atom_style, velocity info includes the translational velocity, angular velocity, and angular momentum of a particle. If the *vel* option is set to *yes*, then ghost atoms store these quantities; if *no* then they do not. The *yes* setting is needed by some pair styles which require the velocity state of both the I and J particles to compute a pairwise I,J interaction.

Note that if the fix deform command is being used with its "remap v" option enabled, then the velocities for ghost atoms (in the fix deform group) mirrored across a periodic boundary will also include components due to any velocity shift that occurs across that boundary (e.g. due to dilation or shear).

Restrictions: none

Related commands:

neighbor

Default:

The default settings are style = single, group = all, cutoff = 0.0, vel = no. The cutoff default of 0.0 means that ghost cutoff = neighbor cutoff = pairwise force cutoff + neighbor skin.

compute atom/molecule command

Syntax:

```
compute ID group-ID atom/molecule input1 input2 ...
```

- ID, group-ID are documented in [compute](#) command
- atom/molecule = style name of this compute command
- one or more inputs can be listed
- input = c_ID, c_ID[N], f_ID, f_ID[N], v_name

```
c_ID = per-atom vector calculated by a compute with ID
c_ID[I] = Ith column of per-atom array calculated by a compute with ID
f_ID = per-atom vector calculated by a fix with ID
f_ID[I] = Ith column of per-atom array calculated by a fix with ID
v_name = per-atom vector calculated by an atom-style variable with name
```

Examples:

```
compute 1 all atom/molecule c_ke c_pe
compute 1 top atom/molecule v_myFormula c_stress3
```

Description:

Define a calculation that sums per-atom values on a per-molecule basis, one per listed input. The inputs can [computes](#), [fixes](#), or [variables](#) that generate per-atom quantities. Note that attributes stored by atoms, such as mass or force, can also be summed on a per-molecule basis, by accessing these quantities via the [compute property/atom](#) command.

Each listed input is operated on independently. Only atoms within the specified group contribute to the per-molecule sum. Note that compute or fix inputs define their own group which may affect the quantities they return. For example, if a compute is used as an input which generates a per-atom vector, it will generate values of 0.0 for atoms that are not in the group specified for that compute.

The ordering of per-molecule quantities produced by this compute is consistent with the ordering produced by other compute commands that generate per-molecule datums. Conceptually, them molecule IDs will be in ascending order for any molecule with one or more of its atoms in the specified group.

If an input begins with "c_", a compute ID must follow which has been previously defined in the input script and which generates per-atom quantities. See the individual [compute](#) doc page for details. If no bracketed integer is appended, the vector calculated by the compute is used. If a bracketed integer is appended, the Ith column of the array calculated by the compute is used. Users can also write code for their own compute styles and [add them to LIGGGHTS(R)-WSU](#).

If an input begins with "f_", a fix ID must follow which has been previously defined in the input script and which generates per-atom quantities. See the individual [fix](#) doc page for details. Note that some fixes only produce their values on certain timesteps, which must be compatible with when compute atom/molecule references the values, else an error results. If no bracketed integer is appended, the vector calculated by the fix is used. If a bracketed integer is appended, the Ith column of the array calculated by the fix is used. Users can also write code for their own fix style and [add them to LIGGGHTS(R)-WSU](#).

If an input begins with "v_", a variable name must follow which has been previously defined in the input script. It must be an [atom-style variable](#). Atom-style variables can reference thermodynamic keywords and

various per-atom attributes, or invoke other computes, fixes, or variables when they are evaluated, so this is a very general means of generating per-atom quantities to sum on a per-molecule basis.

Output info:

This compute calculates a global vector or global array depending on the number of input values. The length of the vector or number of rows in the array is the number of molecules. If a single input is specified, a global vector is produced. If two or more inputs are specified, a global array is produced where the number of columns = the number of inputs. The vector or array can be accessed by any command that uses global values from a compute as input. See [this section](#) for an overview of LIGGGHTS(R)-WSU output options.

All the vector or array values calculated by this compute are "extensive".

The vector or array values will be in whatever [units](#) the input quantities are in.

Restrictions: none

Related commands:

[compute](#), [fix](#), [variable](#)

Default: none

compute bond/local command

Syntax:

```
compute ID group-ID bond/local input1 input2 ...
```

- ID, group-ID are documented in compute command
- bond/local = style name of this compute command
- one or more keywords may be appended
- keyword = *dist* or *eng*

```
  dist = bond distance
    eng = bond energy
  force = bond force
```

Examples:

```
compute 1 all bond/local eng
compute 1 all bond/local dist eng force
```

Description:

Define a computation that calculates properties of individual bond interactions. The number of datums generated, aggregated across all processors, equals the number of bonds in the system, modified by the group parameter as explained below.

The local data stored by this command is generated by looping over all the atoms owned on a processor and their bonds. A bond will only be included if both atoms in the bond are in the specified compute group. Any bonds that have been broken (see the bond_style command) by setting their bond type to 0 are not included. Bonds that have been turned off (see the fix shake or delete_bonds commands) by setting their bond type negative are written into the file, but their energy will be 0.0.

Note that as atoms migrate from processor to processor, there will be no consistent ordering of the entries within the local vector or array from one timestep to the next. The only consistency that is guaranteed is that the ordering on a particular timestep will be the same for local vectors or arrays generated by other compute commands. For example, bond output from the compute property/local command can be combined with data from this command and output by the dump local command in a consistent way.

Here is an example of how to do this:

```
compute 1 all property/local batom1 batom2 btype
compute 2 all bond/local dist eng
dump 1 all local 1000 tmp.dump index c_1[1] c_1[2] c_1[3] c_2[1] c_2[2]
```

Output info:

This compute calculates a local vector or local array depending on the number of keywords. The length of the vector or number of rows in the array is the number of bonds. If a single keyword is specified, a local vector is produced. If two or more keywords are specified, a local array is produced where the number of columns = the number of keywords. The vector or array can be accessed by any command that uses local values from a compute as input. See this section for an overview of LIGGGHTS(R)-WSU output options.

The output for *dist* will be in distance [units](#). The output for *eng* will be in energy [units](#). The output for *force* will be in force [units](#).

Restrictions: none

Related commands:

[dump local](#), [compute property/local](#)

Default: none

compute centro/atom command

Syntax:

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

- ID, group-ID are documented in [compute](#) command
- centro/atom = style name of this compute command
- lattice = *fcc* or *bcc* or N = # of neighbors per atom to include

Examples:

```
compute 1 all centro/atom fcc

compute 1 all centro/atom 8
```

Description:

Define a computation that calculates the centro-symmetry parameter for each atom in the group. In solid-state systems the centro-symmetry parameter is a useful measure of the local lattice disorder around an atom and can be used to characterize whether the atom is part of a perfect lattice, a local defect (e.g. a dislocation or stacking fault), or at a surface.

The value of the centro-symmetry parameter will be 0.0 for atoms not in the specified compute group.

This parameter is computed using the following formula from [(Kelchner)](#)

$$CS = \sum_{i=1}^{N/2} |\vec{R}_i + \vec{R}_{i+N/2}|^2$$

where the *N* nearest neighbors or each atom are identified and Ri and Ri+N/2 are vectors from the central atom to a particular pair of nearest neighbors. There are N*(N-1)/2 possible neighbor pairs that can contribute to this formula. The quantity in the sum is computed for each, and the N/2 smallest are used. This will typically be for pairs of atoms in symmetrically opposite positions with respect to the central atom; hence the i+N/2 notation.

N is an input parameter, which should be set to correspond to the number of nearest neighbors in the underlying lattice of atoms. If the keyword *fcc* or *bcc* is used, *N* is set to 12 and 8 respectively. More generally, *N* can be set to a positive, even integer.

For an atom on a lattice site, surrounded by atoms on a perfect lattice, the centro-symmetry parameter will be 0. It will be near 0 for small thermal perturbations of a perfect lattice. If a point defect exists, the symmetry is broken, and the parameter will be a larger positive value. An atom at a surface will have a large positive parameter. If the atom does not have *N* neighbors (within the potential cutoff), then its centro-symmetry parameter is set to 0.0.

Only atoms within the cutoff of the pairwise neighbor list are considered as possible neighbors. Atoms not in the compute group are included in the *N* neighbors used in this calculation.

The neighbor list needed to compute this quantity is constructed each time the calculation is performed (e.g. each time a snapshot of atoms is dumped). Thus it can be inefficient to compute/dump this quantity too frequently or to have multiple compute/dump commands, each with a *centro/atom* style.

Output info:

This compute calculates a per-atom vector, which can be accessed by any command that uses per-atom values from a compute as input. See Section_howto 15 for an overview of LIGGGHTS(R)-WSU output options.

The per-atom vector values are unitless values >= 0.0. Their magnitude depends on the lattice style due to the number of contributing neighbor pairs in the summation in the formula above. And it depends on the local defects surrounding the central atom, as described above.

Here are typical centro-symmetry values, from a a nanoindentation simulation into gold (FCC). These were provided by Jon Zimmerman (Sandia):

```
Bulk lattice = 0
Dislocation core ~ 1.0 (0.5 to 1.25)
Stacking faults ~ 5.0 (4.0 to 6.0)
Free surface ~ 23.0
```

These values are *not* normalized by the square of the lattice parameter. If they were, normalized values would be:

```
Bulk lattice = 0
Dislocation core ~ 0.06 (0.03 to 0.075)
Stacking faults ~ 0.3 (0.24 to 0.36)
Free surface ~ 1.38
```

For BCC materials, the values for dislocation cores and free surfaces would be somewhat different, due to their being only 8 neighbors instead of 12.

Restrictions: none

Related commands:

compute cna/atom

Default: none

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

compute cluster/atom command

Syntax:

```
compute ID group-ID cluster/atom cutoff
```

- ID, group-ID are documented in **compute** command
- cluster/atom = style name of this compute command
- cutoff = distance within which to label atoms as part of same cluster (distance units)

Examples:

```
compute 1 all cluster/atom 1.0
```

Description:

Define a computation that assigns each atom a cluster ID.

A cluster is defined as a set of atoms, each of which is within the cutoff distance from one or more other atoms in the cluster. If an atom has no neighbors within the cutoff distance, then it is a 1-atom cluster. The ID of every atom in the cluster will be the smallest atom ID of any atom in the cluster.

Only atoms in the compute group are clustered and assigned cluster IDs. Atoms not in the compute group are assigned a cluster ID = 0.

The neighbor list needed to compute this quantity is constructed each time the calculation is performed (i.e. each time a snapshot of atoms is dumped). Thus it can be inefficient to compute/dump this quantity too frequently or to have multiple compute/dump commands, each of a *clsuter/atom* style.

Output info:

This compute calculates a per-atom vector, which can be accessed by any command that uses per-atom values from a compute as input. See [Section_howto 15](#) for an overview of LIGGGHTS(R)-WSU output options.

The per-atom vector values will be an ID > 0, as explained above.

Restrictions: none

Related commands:

[compute coord/atom](#)

Default: none

compute cna/atom command

Syntax:

```
compute ID group-ID cna/atom cutoff
```

- ID, group-ID are documented in [compute](#) command
- cna/atom = style name of this compute command
- cutoff = cutoff distance for nearest neighbors (distance units)

Examples:

```
compute 1 all cna/atom 3.08
```

Description:

Define a computation that calculates the CNA (Common Neighbor Analysis) pattern for each atom in the group. In solid-state systems the CNA pattern is a useful measure of the local crystal structure around an atom. The CNA methodology is described in [(Faken)](#) and [(Tsuzuki)](#).

Currently, there are five kinds of CNA patterns LIGGGHTS(R)-WSU recognizes:

- fcc = 1
- hcp = 2
- bcc = 3
- icosohedral = 4
- unknown = 5

The value of the CNA pattern will be 0 for atoms not in the specified compute group. Note that normally a CNA calculation should only be performed on mono-component systems.

The CNA calculation can be sensitive to the specified cutoff value. You should insure the appropriate nearest neighbors of an atom are found within the cutoff distance for the presumed crystal strucure. E.g. 12 nearest neighbor for perfect FCC and HCP crystals, 14 nearest neighbors for perfect BCC crystals. These formulas can be used to obtain a good cutoff distance:

$$r_c^{fcc} = \frac{1}{2}\left(\frac{\sqrt{2}}{2} + 1\right)a \simeq 0.8536\,a$$

$$r_c^{bcc} = \frac{1}{2}(\sqrt{2} + 1)a \simeq 1.207\,a$$

$$r_c^{hcp} = \frac{1}{2}\left(1 + \sqrt{\frac{4 + 2x^2}{3}}\right)a$$

where a is the lattice constant for the crystal structure concerned and in the HCP case, x = (c/a) / 1.633, where 1.633 is the ideal c/a for HCP crystals.

Also note that since the CNA calculation in LIGGGHTS(R)-WSU uses the neighbors of an owned atom to find the nearest neighbors of a ghost atom, the following relation should also be satisfied:

$$Rc + Rs > 2 * \text{cutoff}$$

where Rc is the cutoff distance of the potential, Rs is the skin distance as specified by the [neighbor](#) command, and cutoff is the argument used with the compute cna/atom command. LIGGGHTS(R)-WSU will issue a warning if this is not the case.

The neighbor list needed to compute this quantity is constructed each time the calculation is performed (e.g. each time a snapshot of atoms is dumped). Thus it can be inefficient to compute/dump this quantity too frequently or to have multiple compute/dump commands, each with a *cna/atom* style.

Output info:

This compute calculates a per-atom vector, which can be accessed by any command that uses per-atom values from a compute as input. See [Section_howto 15](#) for an overview of LIGGGHTS(R)-WSU output options.

The per-atom vector values will be a number from 0 to 5, as explained above.

Restrictions: none

Related commands:

[compute centro/atom](#)

Default: none

(Faken) Faken, Jonsson, Comput Mater Sci, 2, 279 (1994).

(Tsuzuki) Tsuzuki, Branicio, Rino, Comput Phys Comm, 177, 518 (2007).

compute com command

Syntax:

```
compute ID group-ID com
```

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

Examples:

```
compute 1 all com
```

Description:

Define a computation that calculates the center-of-mass of the group of atoms, including all effects due to atoms passing thru periodic boundaries.

A vector of three quantites is calculated by this compute, which are the x,y,z coordinates of the center of mass.

IMPORTANT NOTE: The coordinates of an atom contribute to the center-of-mass in "unwrapped" form, by using the image flags associated with each atom. See the [dump custom](#) command for a discussion of "unwrapped" coordinates. See the Atoms section of the [read_data](#) command for a discussion of image flags and how they are set for each atom. You can reset the image flags (e.g. to 0) before invoking this compute by using the [set image](#) command.

IMPORTANT NOTE: If an atom is part of a rigid body (see the [fix rigid](#) command), it's periodic image flags are altered, and its contribution to the center-of-mass may not reflect its true contribution. See the [fix rigid](#) command for details. Thus, to compute the center-of-mass of rigid bodies as they cross periodic boundaries, you will need to post-process a [dump file](#) containing coordinates of the atoms in the bodies.

Output info:

This compute calculates a global vector of length 3, which can be accessed by indices 1-3 by any command that uses global vector values from a compute as input. See [this section](#) for an overview of LIGGGHTS(R)-WSU output options.

The vector values are "intensive". The vector values will be in distance [units](#).

Restrictions: none

Related commands:

[compute com/molecule](#)

Default: none

compute com/molecule command

Syntax:

```
compute ID group-ID com/molecule
```

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

Examples:

```
compute 1 fluid com/molecule
```

Description:

Define a computation that calculates the center-of-mass of individual molecules. The calculation includes all effects due to atoms passing thru periodic boundaries.

The x,y,z coordinates of the center-of-mass for a particular molecule are only computed if one or more of its atoms are in the specified group. Normally all atoms in the molecule should be in the group, however this is not required. LIGGGHTS(R)-WSU will warn you if this is not the case. Only atoms in the group contribute to the center-of-mass calculation for the molecule.

The ordering of per-molecule quantities produced by this compute is consistent with the ordering produced by other compute commands that generate per-molecule datums. Conceptually, them molecule IDs will be in ascending order for any molecule with one or more of its atoms in the specified group.

IMPORTANT NOTE: The coordinates of an atom contribute to the molecule's center-of-mass in "unwrapped" form, by using the image flags associated with each atom. See the [dump custom](#) command for a discussion of "unwrapped" coordinates. See the Atoms section of the [read_data](#) command for a discussion of image flags and how they are set for each atom. You can reset the image flags (e.g. to 0) before invoking this compute by using the [set image](#) command.

IMPORTANT NOTE: If an atom is part of a rigid body (see the [fix rigid](#) command), it's periodic image flags are altered, and its contribution to the center-of-mass may not reflect its true contribution. See the [fix rigid](#) command for details. Thus, to compute the center-of-mass of rigid bodies as they cross periodic boundaries, you will need to post-process a [dump file](#) containing coordinates of the atoms in the bodies.

Output info:

This compute calculates a global array where the number of rows = Nmolecules and the number of columns = 3 for the x,y,z center-of-mass coordinates of each molecule. These values can be accessed by any command that uses global array values from a compute as input. See [Section_howto 15](#) for an overview of LIGGGHTS(R)-WSU output options.

The array values are "intensive". The array values will be in distance [units](#).

Restrictions: none

Related commands:

compute com

Default: none

compute contact/atom command

Syntax:

```
compute ID group-ID contact/atom
```

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

Examples:

```
compute 1 all contact/atom
```

Description:

Define a computation that calculates the number of contacts for each atom in a group.

The contact number is defined for finite-size spherical particles as the number of neighbor atoms which overlap the central particle, meaning that their distance of separation is less than or equal to the sum of the radii of the two particles.

The value of the contact number will be 0.0 for atoms not in the specified compute group.

Output info:

This compute calculates a per-atom vector, whose values can be accessed by any command that uses per-atom values from a compute as input. See [Section_howto 15](#) for an overview of LIGGGHTS(R)-WSU output options.

The per-atom vector values will be a number >= 0.0, as explained above.

Restrictions:

This compute requires that atoms store a radius as defined by the **[atom_style sphere](#)** command.

Related commands:

[compute coord/atom](#)

Default: none

compute coord/atom command

Syntax:

```
compute ID group-ID coord/atom cutoff keyword value
```

- ID, group-ID are documented in **compute** command
- coord/atom = style name of this compute command cutoff = distance within which to count coordination neighbors (distance units) zero or more keyword/value pairs may be appended to args
- keyword = *mix* or *type1*, *type2*, ...

```
mix value = yes or no -ID
    no = count all neighbors
    yes = count only neighbors that have same atom type

typeN = atom type for Nth coordination count (see asterisk form below)
```

Examples:

```
compute 1 all coord/atom 0.003 mix
compute 1 all coord/atom 2.0
compute 1 all coord/atom 6.0 1 2
compute 1 all coord/atom 6.0 2*4 5*8 *
```

Description:

Define a computation that calculates one or more coordination numbers for each atom in a group.

A coordination number is defined as the number of neighbor atoms with specified atom type(s) that are within the specified cutoff distance from the central atom. Atoms not in the group are included in a coordination number of atoms in the group.

The *typeN* keywords allow you to specify which atom types contribute to each coordination number. One coordination number is computed for each of the *typeN* keywords listed. If no *typeN* keywords are listed, a single coordination number is calculated, which includes atoms of all types (same as the "*" format, see below).

The *typeN* keywords can be specified in one of two ways. An explicit numeric value can be used, as in the 2nd example above. Or a wild-card asterisk can be used to specify a range of atom types. This takes the form "*" or "*n" or "n*" or "m*n". If N = the number of atom types, then an asterisk with no numeric values means all types from 1 to N. A leading asterisk means all types from 1 to n (inclusive). A trailing asterisk means all types from n to N (inclusive). A middle asterisk means all types from m to n (inclusive).

The value of all coordination numbers will be 0.0 for atoms not in the specified compute group.

The neighbor list needed to compute this quantity is constructed each time the calculation is performed (i.e. each time a snapshot of atoms is dumped). Thus it can be inefficient to compute/dump this quantity too frequently.

Keyword *mix* controlls if all neighbors are counted or if only neighbors with same atom type are counted. The latter can be useful to quanitfy mixture of different species.

IMPORTANT NOTE: If you have a bonded system, then the settings of special_bonds command can remove pairwise interactions between atoms in the same bond. This is the default setting for the special_bonds command, and means those pairwise interactions do not appear in the neighbor list. Because this fix uses the neighbor list, it also means those pairs will not be included in the coordination count. One way to get around this, is to write a dump file, and use the rerun command to compute the coordination for snapshots in the dump file. The rerun script can use a special_bonds command that includes all pairs in the neighbor list.

Output info:

If single *type1* keyword is specified (or if none are specified), or the *mix* keyword is used, this compute calculates a per-atom vector. If multiple *typeN* keywords are specified, this compute calculates a per-atom array, with N columns. These values can be accessed by any command that uses per-atom values from a compute as input. See Section_howto 15 for an overview of LIGGGHTS(R)-WSU output options.

The per-atom vector or array values will be a number >= 0.0, as explained above.

Restrictions: none

Related commands:

compute cluster/atom

Default: none

compute coord/gran command

Syntax:

```
compute ID group-ID coord/gran
```

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

Examples:

```
compute 1 all coord/gran
```

Description:

Define a computation that calculates the coordination number for each atom in a group. The value of the coordination number will be 0.0 for atoms not in the specified compute group.

The coordination number is defined as the number of neighbor atoms within the granular cutoff distance from the central atom. The cutoff distance for granular systems is the sum of the radii of the two particles.

The neighbor list needed to compute this quantity is constructed each time the calculation is performed (i.e. each time a snapshot of atoms is dumped). Thus it can be inefficient to compute/dump this quantity too frequently or to have multiple compute/dump commands, each of a coord/gran style.

Output info:

This compute calculates a per-atom vector, which can be accessed by any command that uses per-atom values from a compute as input. See [this section](#) for an overview of LIGGGHTS(R)-WSU output options.

The per-atom vector values will be a number >= 0.0, as explained above.

Restrictions:

This command cannot be applied to multi-sphere simulations, as the output will not be the per-body coordination number.

Related commands: none

Default: none

compute displace/atom command

Syntax:

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

- ID, group-ID are documented in **compute** command
- displace/atom = style name of this compute command

Examples:

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

Description:

Define a computation that calculates the current displacement of each atom in the group from its original coordinates, including all effects due to atoms passing thru periodic boundaries.

A vector of four quantites per atom is calculated by this compute. The first 3 elements of the vector are the dx,dy,dz displacements. The 4th component is the total displacement, i.e. sqrt(dx*dx + dy*dy + dz*dz).

The displacement of an atom is from its original position at the time the compute command was issued. The value of the displacement will be 0.0 for atoms not in the specified compute group.

IMPORTANT NOTE: Initial coordinates are stored in "unwrapped" form, by using the image flags associated with each atom. See the **dump custom** command for a discussion of "unwrapped" coordinates. See the Atoms section of the **read_data** command for a discussion of image flags and how they are set for each atom. You can reset the image flags (e.g. to 0) before invoking this compute by using the **set image** command.

IMPORTANT NOTE: If an atom is part of a rigid body (see the **fix rigid** command), it's periodic image flags are altered, and the computed displacement may not reflect its true displacement. See the **fix rigid** command for details. Thus, to compute the displacement of rigid bodies as they cross periodic boundaries, you will need to post-process a **dump file** containing coordinates of the atoms in the bodies.

IMPORTANT NOTE: If you want the quantities calculated by this compute to be continuous when running from a **restart file**, then you should use the same ID for this compute, as in the original run. This is so that the created fix will also have the same ID, and thus be initialized correctly with atom coordinates from the restart file.

Output info:

This compute calculates a per-atom array with 4 columns, which can be accessed by indices 1-4 by any command that uses per-atom values from a compute as input. See **Section_howto 15** for an overview of LIGGGHTS(R)-WSU output options.

The per-atom array values will be in distance **units**.

Restrictions: none

Related commands:

compute msd, dump custom, fix store/state

Default: none

compute erotate/asphere command

Syntax:

```
compute ID group-ID erotate/asphere
```

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

Examples:

```
compute 1 all erotate/asphere
```

Description:

Define a computation that calculates the rotational kinetic energy of a group of aspherical particles. The aspherical particles can be ellipsoids, or line segments, or triangles. See the [atom_style](#) and [read_data](#) commands for descriptions of these options.

For all 3 types of particles, the rotational kinetic energy is computed as 1/2 I w^2, where I is the inertia tensor for the aspherical particle and w is its angular velocity, which is computed from its angular momentum if needed.

IMPORTANT NOTE: For [2d models](#), ellipsoidal particles are treated as ellipsoids, not ellipses, meaning their moments of inertia will be the same as in 3d.

Output info:

This compute calculates a global scalar (the KE). This value can be used by any command that uses a global scalar value from a compute as input. See [Section_howto 15](#) for an overview of LIGGGHTS(R)-WSU output options.

The scalar value calculated by this compute is "extensive". The scalar value will be in energy [units](#).

Restrictions:

This compute requires that ellipsoidal particles atoms store a shape and quaternion orientation and angular momentum as defined by the [atom_style ellipsoid](#) command.

This compute requires that line segment particles atoms store a length and orientation and angular velocity as defined by the [atom_style line](#) command.

This compute requires that triangular particles atoms store a size and shape and quaternion orientation and angular momentum as defined by the [atom_style tri](#) command.

All particles in the group must be finite-size. They cannot be point particles.

Related commands: none

[compute erotate/sphere](#)

Default: none

compute erotate/multisphere command

Syntax:

```
compute ID group-ID erotate/multisphere
```

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

Examples:

```
compute 1 all erotate/multisphere
```

Description:

Define a computation that calculates the rotational kinetic energy of a collection of multisphere bodies.

The rotational energy of each multisphere body is computed as 1/2 I Wbody^2, where I is the inertia tensor for the multisphere body, and Wbody is its angular velocity vector. Both I and Wbody are in the frame of reference of the multisphere body, i.e. I is diagonalized.

This compute automatically connects to the [fix multisphere](#) commands which defines the multisphere bodies. The group specified in the compute command is ignored. The rotational energy of all the multisphere bodies defined by the fix multisphere command in included in the calculation.

Output info:

This compute calculates a global scalar (the summed rotational energy of all the multisphere bodies). This value can be used by any command that uses a global scalar value from a compute as input. See [Section_howto 15](#) for an overview of LIGGGHTS(R)-WSU output options.

The scalar value calculated by this compute is "extensive". The scalar value will be in energy [units](#).

Restrictions: none

Related commands:

[compute ke/multisphere](#)

Default: none

compute erotate/sphere/atom command

Syntax:

```
compute ID group-ID erotate/sphere/atom
```

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

Examples:

```
compute 1 all erotate/sphere/atom
```

Description:

Define a computation that calculates the rotational kinetic energy for each particle in a group.

The rotational energy is computed as 1/2 I w^2, where I is the moment of inertia for a sphere and w is the particle's angular velocity.

IMPORTANT NOTE: For [2d models,](#) particles are treated as spheres, not disks, meaning their moment of inertia will be the same as in 3d.

The value of the rotational kinetic energy will be 0.0 for atoms not in the specified compute group or for point particles with a radius = 0.0.

Output info:

This compute calculates a per-atom vector, which can be accessed by any command that uses per-atom values from a compute as input. See [Section_howto 15](#) for an overview of LIGGGHTS(R)-WSU output options.

The per-atom vector values will be in energy [units](#).

Restrictions: none

Related commands:

[dump custom](#)

Default: none

compute erotate/sphere command

Syntax:

```
compute ID group-ID erotate/sphere
```

- ID, group-ID are documented in **compute** command
- erotate/sphere = style name of this compute command

Examples:

```
compute 1 all erotate/sphere
```

Description:

Define a computation that calculates the rotational kinetic energy of a group of spherical particles.

The rotational energy is computed as $1/2\, I\, w^2$, where I is the moment of inertia for a sphere and w is the particle's angular velocity.

IMPORTANT NOTE: For [2d models,](#) particles are treated as spheres, not disks, meaning their moment of inertia will be the same as in 3d.

Output info:

This compute calculates a global scalar (the KE). This value can be used by any command that uses a global scalar value from a compute as input. See [Section_howto 15](#) for an overview of LIGGGHTS(R)-WSU output options.

The scalar value calculated by this compute is "extensive". The scalar value will be in energy [units](#).

Restrictions:

This compute requires that atoms store a radius and angular velocity (omega) as defined by the [atom_style sphere](#) command.

All particles in the group must be finite-size spheres or point particles. They cannot be aspherical. Point particles will not contribute to the rotational energy.

Related commands:

[compute erotate/asphere](#)

Default: none

compute group/group command

Syntax:

```
compute ID group-ID group/group group2-ID keyword value ...
```

- ID, group-ID are documented in [compute](#) command
- group/group = style name of this compute command
- group2-ID = group ID of second (or same) group
- zero or more keyword/value pairs may be appended
- keyword = *pair* or *boundary*

```
  pair value = yes or no
    boundary value = yes or no
```

Examples:

```
compute 1 lower group/group upper
compute mine fluid group/group wall
```

Description:

Define a computation that calculates the total energy and force interaction between two groups of atoms: the compute group and the specified group2. The two groups can be the same.

If the *pair* keyword is set to *yes*, which is the default, then the the interaction energy will include a pair component which is defined as the pairwise energy between all pairs of atoms where one atom in the pair is in the first group and the other is in the second group. Likewise, the interaction force calculated by this compute will include the force on the compute group atoms due to pairwise interactions with atoms in the specified group2.

This compute does not calculate any bond interactions between atoms in the two groups.

The pairwise contributions to the group-group interactions are calculated by looping over a neighbor list.

Output info:

This compute calculates a global scalar (the energy) and a global vector of length 3 (force), which can be accessed by indices 1-3. These values can be used by any command that uses global scalar or vector values from a compute as input. See [this section](#) for an overview of LIGGGHTS(R)-WSU output options.

Both the scalar and vector values calculated by this compute are "extensive". The scalar value will be in energy [units](#). The vector values will be in force [units](#).

Restrictions:

Related commands: none

Default:

The option defaults are pair = yes, kspace = no, and boundary = yes.

Bogusz et al, J Chem Phys, 108, 7070 (1998)

compute gyration command

Syntax:

```
compute ID group-ID gyration
```

- ID, group-ID are documented in **compute** command
- gyration = style name of this compute command

Examples:

```
compute 1 molecule gyration
```

Description:

Define a computation that calculates the radius of gyration Rg of the group of atoms, including all effects due to atoms passing thru periodic boundaries.

Rg is a measure of the size of the group of atoms, and is computed by this formula

$$R_g{}^2 = \frac{1}{M} \sum_i m_i (r_i - r_{cm})^2$$

where M is the total mass of the group, Rcm is the center-of-mass position of the group, and the sum is over all atoms in the group.

A Rg tensor, stored as a 6-element vector, is also calculated by this compute. The formula for the components of the tensor is the same as the above formula, except that (Ri - Rcm)^2 is replaced by (Rix - Rcmx) * (Riy - Rcmy) for the xy component, etc. The 6 components of the vector are ordered xx, yy, zz, xy, xz, yz.

IMPORTANT NOTE: The coordinates of an atom contribute to Rg in "unwrapped" form, by using the image flags associated with each atom. See the dump custom command for a discussion of "unwrapped" coordinates. See the Atoms section of the read_data command for a discussion of image flags and how they are set for each atom. You can reset the image flags (e.g. to 0) before invoking this compute by using the set image command.

Output info:

This compute calculates a global scalar (Rg) and a global vector of length 6 (Rg tensor), which can be accessed by indices 1-6. These values can be used by any command that uses a global scalar value or vector values from a compute as input. See Section_howto 15 for an overview of LIGGGHTS(R)-WSU output options.

The scalar and vector values calculated by this compute are "intensive". The scalar and vector values will be in distance units.

Restrictions: none

Related commands:

compute gyration/molecule

Default: none

compute gyration/molecule command

Syntax:

```
compute ID group-ID gyration/molecule keyword value ...
```

- ID, group-ID are documented in [compute](#) command
- gyration/molecule = style name of this compute command
- zero or more keyword/value pairs may be appended
- keyword = *tensor*

  tensor value = none

Examples:

```
compute 1 molecule gyration/molecule
compute 2 molecule gyration/molecule tensor
```

Description:

Define a computation that calculates the radius of gyration Rg of individual molecules. The calculation includes all effects due to atoms passing thru periodic boundaries.

Rg is a measure of the size of a molecule, and is computed by this formula

$$R_g{}^2 = \frac{1}{M} \sum_i m_i (r_i - r_{cm})^2$$

where M is the total mass of the molecule, Rcm is the center-of-mass position of the molecule, and the sum is over all atoms in the molecule and in the group.

If the *tensor* keyword is specified, then the scalar Rg value is not calculated, but an Rg tensor is instead calculated for each molecule. The formula for the components of the tensor is the same as the above formula, except that (Ri - Rcm)^2 is replaced by (Rix - Rcmx) * (Riy - Rcmy) for the xy component, etc. The 6 components of the tensor are ordered xx, yy, zz, xy, xz, yz.

Rg for a particular molecule is only computed if one or more of its atoms are in the specified group. Normally all atoms in the molecule should be in the group, however this is not required. LIGGGHTS(R)-WSU will warn you if this is not the case. Only atoms in the group contribute to the Rg calculation for the molecule.

The ordering of per-molecule quantities produced by this compute is consistent with the ordering produced by other compute commands that generate per-molecule datums. Conceptually, them molecule IDs will be in ascending order for any molecule with one or more of its atoms in the specified group.

IMPORTANT NOTE: The coordinates of an atom contribute to Rg in "unwrapped" form, by using the image flags associated with each atom. See the [dump custom](#) command for a discussion of "unwrapped" coordinates. See the Atoms section of the [read_data](#) command for a discussion of image flags and how they are set for each atom. You can reset the image flags (e.g. to 0) before invoking this compute by using the [set image](#) command.

Output info:

This compute calculates a global vector if the *tensor* keyword is not specified and a global array if it is. The length of the vector or number of rows in the array is the number of molecules. If the *tensor* keyword is specified, the global array has 6 columns. The vector or array can be accessed by any command that uses global values from a compute as input. See [this section](#) for an overview of LIGGGHTS(R)-WSU output options.

All the vector or array values calculated by this compute are "intensive". The vector or array values will be in distance [units](#).

Restrictions: none

Related commands: none

[compute gyration](#)

Default: none

compute command

Syntax:

```
compute ID group-ID style args
```

- ID = user-assigned name for the computation
- group-ID = ID of the group of atoms to perform the computation on
- style = one of a list of possible style names (see below)
- args = arguments used by a particular style

Examples:

```
compute 1 all temp
compute newtemp flow temp/partial 1 1 0
compute 3 all ke/atom
```

Description:

Define a computation that will be performed on a group of atoms. Quantities calculated by a compute are instantaneous values, meaning they are calculated from information about atoms on the current timestep or iteration, though a compute may internally store some information about a previous state of the system. Defining a compute does not perform a computation. Instead computes are invoked by other LIGGGHTS(R)-WSU commands as needed, e.g. to calculate dump file output. See this howto section for a summary of various LIGGGHTS(R)-WSU output options, many of which involve computes.

The full list of fixes defined in LIGGGHTS(R)-WSU is on this page.

The ID of a compute can only contain alphanumeric characters and underscores.

Computes calculate one of three styles of quantities: global, per-atom, or local. A global quantity is one or more system-wide values, e.g. the temperature of the system. A per-atom quantity is one or more values per atom, e.g. the kinetic energy of each atom. Per-atom values are set to 0.0 for atoms not in the specified compute group. Local quantities are calculated by each processor based on the atoms it owns, but there may be zero or more per atom, e.g. a list of bond distances. Computes that produce per-atom quantities have the word "atom" in their style, e.g. *ke/atom*. Computes that produce local quantities have the word "local" in their style, e.g. *bond/local*. Styles with neither "atom" or "local" in their style produce global quantities.

Note that a single compute produces either global or per-atom or local quantities, but never more than one of these.

Global, per-atom, and local quantities each come in three kinds: a single scalar value, a vector of values, or a 2d array of values. The doc page for each compute describes the style and kind of values it produces, e.g. a per-atom vector. Some computes produce more than one kind of a single style, e.g. a global scalar and a global vector.

When a compute quantity is accessed, as in many of the output commands discussed below, it can be referenced via the following bracket notation, where ID is the ID of the compute:

c_ID	entire scalar, vector, or array
c_ID[I]	one element of vector, one column of array

c_ID[I][J]	one element of array

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

Note that commands and variables which use compute quantities typically do not allow for all kinds, e.g. a command may require a vector of values, not a scalar. This means there is no ambiguity about referring to a compute quantity as c_ID even if it produces, for example, both a scalar and vector. The doc pages for various commands explain the details.

In LIGGGHTS(R)-WSU, the values generated by a compute can be used in several ways:

- Global values can be output via the thermo_style custom or fix ave/time command. Or the values can be referenced in a variable equal or variable atom command.
- Per-atom values can be output via the dump custom command or the fix ave/spatial command. Or they can be time-averaged via the fix ave/atom command or reduced by the compute reduce command. Or the per-atom values can be referenced in an atom-style variable.
- Local values can be reduced by the compute reduce command, or histogrammed by the fix ave/histo command, or output by the dump local command.

The results of computes that calculate global quantities can be either "intensive" or "extensive" values. Intensive means the value is independent of the number of atoms in the simulation, e.g. temperature. Extensive means the value scales with the number of atoms in the simulation, e.g. total rotational kinetic energy. Thermodynamic output will normalize extensive values by the number of atoms in the system, depending on the "thermo_modify norm" setting. It will not normalize intensive values. If a compute value is accessed in another way, e.g. by a variable, you may want to know whether it is an intensive or extensive value. See the doc page for individual computes for further info.

Properties of either a default or user-defined compute can be modified via the compute_modify command.

Computes can be deleted with the uncompute command.

Code for new computes can be added to LIGGGHTS(R)-WSU (see this section of the manual) and the results of their calculations accessed in the various ways described above.

Each compute style has its own doc page which describes its arguments and what it does. The full list of computes defined in LIGGGHTS(R)-WSU is on this page.

Restrictions: none

Related commands:

uncompute, compute_modify, fix ave/atom, fix ave/spatial, fix ave/time, fix ave/histo

Default: none

compute inertia/molecule command

Syntax:

```
compute ID group-ID inertia/molecule
```

- ID, group-ID are documented in **compute** command
- inertia/molecule = style name of this compute command

Examples:

```
compute 1 fluid inertia/molecule
```

Description:

Define a computation that calculates the inertia tensor of individual molecules. The calculation includes all effects due to atoms passing thru periodic boundaries.

The symmetric intertia tensor has 6 components, ordered Ixx,Iyy,Izz,Ixy,Iyz,Ixz. The tensor for a particular molecule is only computed if one or more of its atoms is in the specified group. Normally all atoms in the molecule should be in the group, however this is not required. LIGGGHTS(R)-WSU will warn you if this is not the case. Only atoms in the group contribute to the inertia tensor and associated center-of-mass calculation for the molecule.

The ordering of per-molecule quantities produced by this compute is consistent with the ordering produced by other compute commands that generate per-molecule datums. Conceptually, the molecule IDs will be in ascending order for any molecule with one or more of its atoms in the specified group.

IMPORTANT NOTE: The coordinates of an atom contribute to the molecule's inertia tensor in "unwrapped" form, by using the image flags associated with each atom. See the **dump custom** command for a discussion of "unwrapped" coordinates. See the Atoms section of the **read_data** command for a discussion of image flags and how they are set for each atom. You can reset the image flags (e.g. to 0) before invoking this compute by using the **set image** command.

IMPORTANT NOTE: If an atom is part of a rigid body (see the **fix rigid** command), it's periodic image flags are altered, and its contribution to the inertia tensor may not reflect its true contribution. See the **fix rigid** command for details. Thus, to compute the inertia tensor of rigid bodies as they cross periodic boundaries, you will need to post-process a **dump file** containing coordinates of the atoms in the bodies.

Output info:

This compute calculates a global array where the number of rows = Nmolecules and the number of columns = 6 for the 6 components of the inertia tensor of each molecule, ordered as listed above. These values can be accessed by any command that uses global array values from a compute as input. See **Section_howto 15** for an overview of LIGGGHTS(R)-WSU output options.

The array values are "intensive". The array values will be in distance **units**.

Restrictions: none

Related commands:

compute inertia/molecule command   55

[variable inertia() function](#)

Default: none

compute ke/atom command

Syntax:

```
compute ID group-ID ke/atom
```

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

Examples:

```
compute 1 all ke/atom
```

Description:

Define a computation that calculates the per-atom translational kinetic energy for each atom in a group.

The kinetic energy is simply $1/2\ m\ v^2$, where m is the mass and v is the velocity of each atom.

The value of the kinetic energy will be 0.0 for atoms not in the specified compute group.

Output info:

This compute calculates a per-atom vector, which can be accessed by any command that uses per-atom values from a compute as input. See [Section_howto 15](#) for an overview of LIGGGHTS(R)-WSU output options.

The per-atom vector values will be in energy [units](#).

Restrictions: none

Related commands:

[dump custom](#)

Default: none

compute ke command

Syntax:

```
compute ID group-ID ke
```

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

Examples:

```
compute 1 all ke
```

Description:

Define a computation that calculates the translational kinetic energy of a group of particles.

The kinetic energy of each particle is computed as 1/2 m v^2, where m and v are the mass and velocity of the particle.

There is a subtle difference between the quantity calculated by this compute and the kinetic energy calculated by the *ke* keyword used in thermodynamic output, as specified by the [thermo_style](#) command. For this compute, kinetic energy is "translational" kinetic energy, calculated by the simple formula above. For thermodynamic output, the *ke* keyword infers kinetic energy from the temperature of the system with 1/2 Kb T of energy for each degree of freedom. For the default temperature computation via the [compute temp](#) command, these are the same. But different computes that calculate temperature can subtract out different non-thermal components of velocity and/or include different degrees of freedom (translational, rotational, etc).

Output info:

This compute calculates a global scalar (the summed KE). This value can be used by any command that uses a global scalar value from a compute as input. See [Section_howto 15](#) for an overview of LIGGGHTS(R)-WSU output options.

The scalar value calculated by this compute is "extensive". The scalar value will be in energy [units](#).

Restrictions: none

Related commands:

[compute erotate/sphere](#)

Default: none

compute ke/multisphere command

Syntax:

```
compute ID group-ID ke/multisphere
```

- ID, group-ID are documented in **compute** command
- ke = style name of this compute command

Examples:

```
compute 1 all ke/multisphere
```

Description:

Define a computation that calculates the translational kinetic energy of a collection of multisphere bodies.

The kinetic energy of each multisphere body is computed as 1/2 M Vcm^2, where M is the total mass of the multisphere body, and Vcm is its center-of-mass velocity.

This compute automatically connects to the [fix multisphere](#) commands which defines the multisphere bodies. The group specified in the compute command is ignored. The kinetic energy of all the multisphere bodies defined by the fix multisphere command in included in the calculation.

Output info:

This compute calculates a global scalar (the summed KE of all the multisphere bodies). This value can be used by any command that uses a global scalar value from a compute as input. See [Section_howto 15](#) for an overview of LIGGGHTS(R)-WSU output options.

The scalar value calculated by this compute is "extensive". The scalar value will be in energy [units](#).

Restrictions: none

Related commands:

[compute erotate/multisphere](#)

Default: none

compute_modify command

Syntax:

```
compute_modify compute-ID keyword value ...
```

- compute-ID = ID of the compute to modify
- one or more keyword/value pairs may be listed
- keyword = *extra* or *dynamic*

```
  extra value = N
      N = # of extra degrees of freedom to subtract
    dynamic value = yes or no
      yes/no = do or do not recompute the number of atoms contributing to the temperature
    thermo value = yes or no
      yes/no = do or do not add contributions from fixes to the potential energy
```

Examples:

```
compute_modify myTemp extra 0
compute_modify newtemp dynamic yes extra 600
```

Description:

Modify one or more parameters of a previously defined compute. Not all compute styles support all parameters.

The *extra* keyword refers to how many degrees-of-freedom are subtracted (typically from 3N) as a normalizing factor in a temperature computation. Only computes that compute a temperature use this option. The default is 2 or 3 for [2d or 3d systems](#) which is a correction factor for an ensemble of velocities with zero total linear momentum. You can use a negative number for the *extra* parameter if you need to add degrees-of-freedom. See the [compute temp/asphere](#) command for an example.

The *dynamic* keyword determines whether the number of atoms N in the compute group is re-computed each time a temperature is computed. Only compute styles that compute a temperature use this option. By default, N is assumed to be constant. If you are adding atoms to the system (see the [fix pour](#) or [fix deposit](#) commands) or expect atoms to be lost (e.g. due to evaporation), then this option can be used to insure the temperature is correctly normalized.

The *thermo* keyword determines whether the potential energy contribution calculated by some [fixes](#) is added to the potential energy calculated by the compute. Currently, only the compute of style *pe* uses this option. See the doc pages for [individual fixes](#) for details.

Restrictions: none

Related commands:

[compute](#)

Default:

The option defaults are extra = 2 or 3 for 2d or 3d systems and dynamic = no. Thermo is *yes* if the compute of style *pe* was defined with no extra keywords; otherwise it is *no*.

compute msd command

Syntax:

```
compute ID group-ID msd keyword values ...
```

- ID, group-ID are documented in [compute](#) command
- msd = style name of this compute command
- zero or more keyword/value pairs may be appended
- keyword = *com*

```
  com value = yes or no
```

Examples:

```
compute 1 all msd
compute 1 upper msd com yes
```

Description:

Define a computation that calculates the mean-squared displacement (MSD) of the group of atoms, including all effects due to atoms passing thru periodic boundaries. For computation of the non-Gaussian parameter of mean-squared displacement, see the [compute msd/nongauss](#) command.

A vector of four quantites is calculated by this compute. The first 3 elements of the vector are the squared dx,dy,dz displacements, summed and averaged over atoms in the group. The 4th element is the total squared displacement, i.e. (dx*dx + dy*dy + dz*dz), summed and averaged over atoms in the group.

The slope of the mean-squared displacement (MSD) versus time is proportional to the diffusion coefficient of the diffusing atoms.

The displacement of an atom is from its original position at the time the compute command was issued. The value of the displacement will be 0.0 for atoms not in the specified compute group.

If the *com* option is set to *yes* then the effect of any drift in the center-of-mass of the group of atoms is subtracted out before the displacment of each atom is calcluated.

IMPORTANT NOTE: Initial coordinates are stored in "unwrapped" form, by using the image flags associated with each atom. See the [dump custom](#) command for a discussion of "unwrapped" coordinates. See the Atoms section of the [read_data](#) command for a discussion of image flags and how they are set for each atom. You can reset the image flags (e.g. to 0) before invoking this compute by using the [set image](#) command.

IMPORTANT NOTE: If an atom is part of a rigid body (see the [fix rigid](#) command), it's periodic image flags are altered, and its contribution to the MSD may not reflect its true contribution. See the [fix rigid](#) command for details. Thus, to compute the MSD of rigid bodies as they cross periodic boundaries, you will need to post-process a [dump file](#) containing coordinates of the atoms in the bodies.

IMPORTANT NOTE: If you want the quantities calculated by this compute to be continuous when running from a [restart file](#), then you should use the same ID for this compute, as in the original run. This is so that the created fix will also have the same ID, and thus be initialized correctly with atom coordinates from the restart file.

Output info:

This compute calculates a global vector of length 4, which can be accessed by indices 1-4 by any command that uses global vector values from a compute as input. See this section for an overview of LIGGGHTS(R)-WSU output options.

The vector values are "intensive". The vector values will be in distance^2 units.

Restrictions: none

Related commands:

compute msd/nongauss, compute displace_atom, fix store/state, compute msd/molecule

Default:

The option default is com = no.

compute msd/molecule command

Syntax:

```
compute ID group-ID msd/molecule
```

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

Examples:

```
compute 1 all msd/molecule
```

Description:

Define a computation that calculates the mean-squared displacement (MSD) of individual molecules. The calculation includes all effects due to atoms passing thru periodic boundaries.

Four quantites are calculated by this compute for each molecule. The first 3 quantities are the squared dx,dy,dz displacements of the center-of-mass. The 4th component is the total squared displacement, i.e. (dx*dx + dy*dy + dz*dz) of the center-of-mass.

The slope of the mean-squared displacement (MSD) versus time is proportional to the diffusion coefficient of the diffusing molecules.

The displacement of the center-of-mass of the molecule is from its original center-of-mass position at the time the compute command was issued.

The MSD for a particular molecule is only computed if one or more of its atoms are in the specified group. Normally all atoms in the molecule should be in the group, however this is not required. LIGGGHTS(R)-WSU will warn you if this is not the case. Only atoms in the group contribute to the center-of-mass calculation for the molecule, which is used to caculate its initial and current position.

The ordering of per-molecule quantities produced by this compute is consistent with the ordering produced by other compute commands that generate per-molecule datums. Conceptually, them molecule IDs will be in ascending order for any molecule with one or more of its atoms in the specified group.

IMPORTANT NOTE: The initial coordinates of each molecule are stored in "unwrapped" form, by using the image flags associated with each atom. See the [dump custom](#) command for a discussion of "unwrapped" coordinates. See the Atoms section of the [read_data](#) command for a discussion of image flags and how they are set for each atom. You can reset the image flags (e.g. to 0) before invoking this compute by using the [set image](#) command.

IMPORTANT NOTE: If an atom is part of a rigid body (see the [fix rigid](#) command), it's periodic image flags are altered, and its contribution to the MSD may not reflect its true contribution. See the [fix rigid](#) command for details. Thus, to compute the MSD of rigid bodies as they cross periodic boundaries, you will need to post-process a [dump file](#) containing coordinates of the atoms in the bodies.

IMPORTANT NOTE: Unlike the [compute msd](#) command, this compute does not store the initial center-of-mass coorindates of its molecules in a restart file. Thus you cannot continue the MSD per molecule calculation of this compute when running from a [restart file](#).

Output info:

This compute calculates a global array where the number of rows = Nmolecules and the number of columns = 4 for dx,dy,dz and the total displacement. These values can be accessed by any command that uses global array values from a compute as input. See [this section](#) for an overview of LIGGGHTS(R)-WSU output options.

The array values are "intensive". The array values will be in distance^2 [units](#).

Restrictions: none

Related commands:

[compute msd](#)

Default: none

compute msd/nongauss command

Syntax:

```
compute ID group-ID msd/nongauss keyword values ...
```

- ID, group-ID are documented in [compute](#) command
- msd/nongauss = style name of this compute command
- zero or more keyword/value pairs may be appended
- keyword = *com*

    com value = *yes* or *no*

Examples:

```
compute 1 all msd/nongauss
compute 1 upper msd/nongauss com yes
```

Description:

Define a computation that calculates the mean-squared displacement (MSD) and non-Gaussian parameter (NGP) of the group of atoms, including all effects due to atoms passing thru periodic boundaries.

A vector of three quantites is calculated by this compute. The first element of the vector is the total squared dx,dy,dz displacements drsquared = (dx*dx + dy*dy + dz*dz) of atoms, and the second is the fourth power of these displacements drfourth = (dx*dx + dy*dy + dz*dz)*(dx*dx + dy*dy + dz*dz), summed and averaged over atoms in the group. The 3rd component is the nonGaussian diffusion paramter NGP = 3*drfourth/(5*drsquared*drsquared), i.e.

$$NGP(t) = 3 < (r(t) - r(0))^4 > /(5 < (r(t) - r(0))^2 >^2) - 1$$

The NGP is a commonly used quantity in studies of dynamical heterogeneity. Its minimum theoretical value (-0.4) occurs when all atoms have the same displacement magnitude. NGP=0 for Brownian diffusion, while NGP > 0 when some mobile atoms move faster than others.

If the *com* option is set to *yes* then the effect of any drift in the center-of-mass of the group of atoms is subtracted out before the displacment of each atom is calcluated.

See the [compute msd](#) doc page for further IMPORTANT NOTES, which also apply to this compute.

Output info:

This compute calculates a global vector of length 3, which can be accessed by indices 1-3 by any command that uses global vector values from a compute as input. See [this section](#) for an overview of LIGGGHTS(R)-WSU output options.

The vector values are "intensive". The first vector value will be in distance^2 [units](#), the second is in distance^4 units, and the 3rd is dimensionless.

Restrictions:

This compute is part of the MISC package. It is only enabled if LIGGGHTS(R)-WSU was built with that package. See the [Making LIGGGHTS(R)-WSU](#) section for more info.

Related commands:

[compute msd](#)

Default:

The option default is com = no.

compute nparticles/tracer/region command

Syntax:

```
compute ID group-ID nparticles/tracer/region
```

- ID, group-ID are documented in compute command
- nparticles/tracer/region = style name of this compute command
- region_count = obligatory keyword
- region-ID = ID of region atoms must be in to be counted
- tracer = obligatory keyword
- tracer-ID = ID of a fix of type fix property/atom/tracer
- zero or more keyword/value pairs may be appended to args
- keyword = *periodic* or *reset_marker*

```
periodic value = dim image
    dim = x or y or z
    image = image that a particle has to be in to be counted (any integer number or all)
  reset_marker value = yes or no
    yes = un-mark particles after counting them
    no = do not un-mark particles after counting them
```

Examples:

```
compute nparticles all nparticles/tracer/region region_count count tracer tr periodic z -1
```

Description:

Define a computation that calculates the number and mass of marked and un-marked particles that are in the region speficied via the *region_count* keyword. Particles have to be in the group "group-ID" to be counted.

Note that only particles marked by a fix property/atom/tracer or fix property/atom/tracer/stream command are counted - therefore, a valid ID of such a fix has to be provided via the *tracer* keyword.

The *reset_marker* keyword controls if particles are un-marked (default) after they have been counted once by this command.

IMPORTANT NOTE: If multiple compute nparticles/tracer/region commands are operating on the same fix property/atom/tracer commands, and the first compute resets the marker value, the second compute will not count them.

With the *periodic* keyword, you can restrict counting/unmarking to particles which are in a specified image in a periodic simulation. For example, using

```
periodic z +2
```

means that particles are only counted if they are in z-image #2. By default, all particles are counted/unmarked regardless in which periodic image they are.

IMPORTANT NOTE: Currently, this command only supports one periodic boundary restriction via the *periodic* keyword. If keyword *periodic* is used multiple times, the last setting will be applied.

Output info:

This this compute calculates a global vector containing the following information (the number in brackets corresponds to the vector id):

- **(1)** total number of (marked + un-marked) particles in region
- **(2)** number of marked particles in region
- **(3)** total mass of (marked + un-marked) particles in region
- **(4)** mass of marked particles in region

See this section for an overview of LIGGGHTS(R)-WSU output options.

Restrictions:

Currently, only one periodic restriction via the *periodic* keyword can be used.

Related commands:

fix property/atom/tracer

Default: *reset_marker* = yes, *periodic* is off per default

compute pair/gran/local command

compute wall/gran/local command

Syntax:

```
compute ID group-ID pair/gran/local keywords ...
compute ID group-ID wall/gran/local keywords ...
```

- ID, group-ID are documented in compute command
- pair/gran/local or wall/gran/local = style name of this compute command
- zero or more keywords may be appended

```
keyword = pos or vel or id or force or torque or history or contactArea or delta:l
  pos = positions of particles in contact (6 values)
  vel = velocities of particles in contact (6 values)
  id = IDs of particles in contact and a periodicity flag (3 values) or IDs of the mesh, th
  force = contact force (3 values)
  force_normal = normal component of contact force (3 values)
  force_tangential = tangential component of contact force (3 values)
  torque = torque divided by particle diameter (3 values)
  history = contact history (# depends on pair style, e.g. 3 shear history values)
  contactArea = area of the contact (1 value)
  delta = overlap of the contact (1 value)
  heatFlux = conductive heat flux of the contact (1 value)
```

Examples:

```
compute 1 all pair/gran/local
compute 1 all pair/gran/local pos force
compute 1 all wall/gran/local
```

Description:

Define a computation that calculates properties of individual pairwise or particle-wall interactions of a granular pair style. The number of datums generated, aggregated across all processors, equals the number of pairwise interactions or particle-wall interactions in the system.

The local data stored by this command is generated by looping over the pairwise neighbor list. Info about an individual pairwise interaction will only be included if both atoms in the pair are in the specified compute group, and if the current pairwise distance is less than the force cutoff distance for that interaction, as defined by the pair_style and pair_coeff commands.

IMPORTANT NOTE: For accessing particle-wall contact data, only mesh walls (see fix mesh) can be used. For computing particle-wall (compute wall/gran/local), the code will automatically look for a fix wall/gran command that uses mesh walls. The order of the meshes in the fix wall/gran command is called the mesh id (starting with 0), and the triangle id reflects the order of the triangles in the STL/VTK file read via the dedicated fix mesh command. For how to output the trangle id, see "dump mesh/gran/VTK command"dump.html.

The output *pos* is the particle positions (6 values) in distance units. Keyword *vel* will do the same for velocities. For computing pairwise data, the output *id* will be the two particle IDs (using this option requires to use an atom map) and a flag that is 1 for interaction over a periodic boundary and 0 otherwise. For computing particle-wall data, the output *id* will be the mesh id, the triangle id and the particle id. The output

force, *force_normal*, *force_tangential* and *torque* are the total contact force, the normal and tangential components of the contact force, and the torque divided by the particle radius, both in force units. Note that the torque does NOT contain any rolling friction torque. The output *history* will depend on what this history represents, according to the granular pair style used. The output *contactArea* will output the contact area, in distance^2 units. Note that *contactArea* is based on an analytic geometric calculation of sphere-sphere or sphere-plane intersection rather than a calculation based on mechanics. This is to ensure that *contactArea* works with all types of contact models.

The output *delta* will output the overlap (sum of radii - distance between particle centers) in distance units .The output *heatFlux* (available only if a fix heat/gran is used to compute heat fluxes) will output the per-contact conductive heat flux area, in energy/time units.

IMPORTANT NOTE: The data associated to the different keywords is output in the following order: *pos*, *vel*, *id*, *force*, *force_normal*, *force_tangential*, *torque*, *history*, *contactArea*, *heatFlux*. This is independant of the order in which the keywords are specified.

Note that as atoms migrate from processor to processor, there will be no consistent ordering of the entries within the local vector or array from one timestep to the next. The only consistency that is guaranteed is that the ordering on a particular timestep will be the same for local vectors or arrays generated by other compute commands. For example, pair output from the compute property/local command can be combined with data from this command and output by the dump local command in a consistent way.

Output info:

This compute calculates a local vector or local array depending on the number of keywords. The length of the vector or number of rows in the array is the number of pairs. If a single keyword is specified, a local vector is produced. If two or more keywords are specified, a local array is produced where the number of columns = the number of keywords. The vector or array can be accessed by any command that uses local values from a compute as input. See this section for an overview of LIGGGHTS(R)-WSU output options.

For information on the units of the output, see above.

Restrictions:

Can only be used together with a granular pair style. For accessing particle-wall contact data, only mesh walls can be used.

Related commands:

dump local, compute property/local

Default:

By default, all of the outputs keywords (except force_normal, force_tangential, heat flux and delta) are activated, i.e. when no keyword is used, positions velocities, ids, forces, torques, history and contact area are output.

compute pe/atom command

Syntax:

```
compute ID group-ID pe/atom keyword ...
```

- ID, group-ID are documented in compute command
- pe/atom = style name of this compute command
- zero or more keywords may be appended
- keyword = *pair* or *bond* or *angle* or *dihedral* or *improper* or *kspace*

Examples:

```
compute 1 all pe/atom
compute 1 all pe/atom pair
compute 1 all pe/atom pair bond
```

Description:

Define a computation that computes the per-atom potential energy for each atom in a group. See the compute pe command if you want the potential energy of the entire system.

The per-atom energy is calculated by the various pair, bond, etc potentials defined for the simulation. If no extra keywords are listed, then the potential energy is the sum of pair, bond, angle, dihedral,improper, and kspace energy. If any extra keywords are listed, then only those components are summed to compute the potential energy.

Note that the energy of each atom is due to its interaction with all other atoms in the simulation, not just with other atoms in the group.

For an energy contribution produced by a small set of atoms (e.g. 4 atoms in a dihedral or 3 atoms in a Tersoff 3-body interaction), that energy is assigned in equal portions to each atom in the set. E.g. 1/4 of the dihedral energy to each of the 4 atoms.

The dihedral_style charmm style calculates pairwise interactions between 1-4 atoms. The energy contribution of these terms is included in the pair energy, not the dihedral energy.

The KSpace contribution is calculated using the method in (Heyes) for the Ewald method and a related method for PPPM, as specified by the kspace_style pppm command. For PPPM, the calcluation requires 1 extra FFT each timestep that per-atom energy is calculated. Thie document describes how the long-range per-atom energy calculation is performed.

As an example of per-atom potential energy compared to total potential energy, these lines in an input script should yield the same result in the last 2 columns of thermo output:

```
compute        peratom all pe/atom
compute        pe all reduce sum c_peratom
thermo_style   custom step temp etotal press pe c_pe
```

IMPORTANT NOTE: The per-atom energy does not any Lennard-Jones tail corrections invoked by the pair_modify tail yes command, since those are global contributions to the system energy.

Output info:

This compute calculates a per-atom vector, which can be accessed by any command that uses per-atom values from a compute as input. See Section_howto 15 for an overview of LIGGGHTS(R)-WSU output options.

The per-atom vector values will be in energy units.

Restrictions:

This compute does not include the potential energy due to the overlap of granular particles.

Related commands:

compute pe, compute stress/atom

Default: none

(Heyes) Heyes, Phys Rev B 49, 755 (1994),

compute pe command

Syntax:

```
compute ID group-ID pe keyword ...
```

- ID, group-ID are documented in [compute](#) command
- pe = style name of this compute command
- zero or more keywords may be appended
- keyword = *pair* or *bond* or *angle* or *dihedral* or *improper* or *kspace*

Examples:

```
compute 1 all pe
compute molPE all pe bond angle dihedral improper
```

Description:

Define a computation that calculates the potential energy of the entire system of atoms. The specified group must be "all". See the [compute pe/atom](#) command if you want per-atom energies. These per-atom values could be summed for a group of atoms via the [compute reduce](#) command.

The energy is calculated by the various pair, bond, etc potentials defined for the simulation. If no extra keywords are listed, then the potential energy is the sum of pair, bond, angle, dihedral, improper, and kspace (long-range) energy. If any extra keywords are listed, then only those components are summed to compute the potential energy.

Various fixes can contribute to the total potential energy of the system. See the doc pages for [individual fixes](#) for details. The *thermo* option of the [compute_modify](#) command determines whether these contributions are added into the computed potential energy. If no keywords are specified the default is *yes*. If any keywords are specified, the default is *no*.

Output info:

This compute calculates a global scalar (the potential energy). This value can be used by any command that uses a global scalar value from a compute as input. See [Section_howto 15](#) for an overview of LIGGGHTS(R)-WSU output options.

The scalar value calculated by this compute is "extensive". The scalar value will be in energy [units](#).

Restrictions:

This compute does not include the potential energy due to the overlap of granular particles.

Related commands:

[compute pe/atom](#)

Default: none

compute pressure command

Syntax:

```
compute ID group-ID pressure temp-ID keyword ...
```

- ID, group-ID are documented in compute command
- pressure = style name of this compute command
- temp-ID = ID of compute that calculates temperature
- zero or more keywords may be appended
- keyword = *ke* or *pair* or *bond* or *angle* or *dihedral* or *improper* or *kspace* or *fix* or *virial*

Examples:

```
compute 1 all pressure myTemp
compute 1 all pressure thermo_temp pair bond
```

Description:

Define a computation that calculates the pressure of the entire system of atoms. The specified group must be "all". See the compute stress/atom command if you want per-atom pressure (stress). These per-atom values could be summed for a group of atoms via the compute reduce command.

The pressure is computed by the formula

$$P = \frac{Nk_BT}{V} + \frac{\sum_i^N r_i \bullet f_i}{dV}$$

where N is the number of atoms in the system (see discussion of DOF below), Kb is the Boltzmann constant, T is the temperature, d is the dimensionality of the system (2 or 3 for 2d/3d), V is the system volume (or area in 2d), and the second term is the virial, computed within LIGGGHTS(R)-WSU for all pairwise as well as 2-body, 3-body, and 4-body, and long-range interactions. Fixes that impose constraints (e.g. the fix shake command) also contribute to the virial term.

A symmetric pressure tensor, stored as a 6-element vector, is also calculated by this compute. The 6 components of the vector are ordered xx, yy, zz, xy, xz, yz. The equation for the I,J components (where I and J = x,y,z) is similar to the above formula, except that the first term uses components of the kinetic energy tensor and the second term uses components of the virial tensor:

$$P_{IJ} = \frac{\sum_k^N m_k v_{k_I} v_{k_J}}{V} + \frac{\sum_k^N r_{k_I} f_{k_J}}{V}$$

If no extra keywords are listed, the entire equations above are calculated which include a kinetic energy (temperature) term and the virial as the sum of pair, bond, angle, dihedral, improper, kspace (long-range), and

fix contributions to the force on each atom. If any extra keywords are listed, then only those components are summed to compute temperature or ke and/or the virial. The *virial* keyword means include all terms except the kinetic energy *ke*.

The temperature and kinetic energy tensor is not calculated by this compute, but rather by the temperature compute specified with the command. Normally this compute should calculate the temperature of all atoms for consistency with the virial term, but any compute style that calculates temperature can be used, e.g. one that excludes frozen atoms or other degrees of freedom.

Note that the N in the first formula above is really degrees-of-freedom divided by d = dimensionality, where the DOF value is calcluated by the temperature compute. See the various compute temperature styles for details.

A compute of this style with the ID of "thermo_press" is created when LIGGGHTS(R)-WSU starts up, as if this command were in the input script:

```
compute thermo_press all pressure thermo_temp
```

where "thermo_temp" is the ID of a similarly defined compute of style "temp". See the "thermo_style" command for more details.

Output info:

This compute calculates a global scalar (the pressure) and a global vector of length 6 (pressure tensor), which can be accessed by indices 1-6. These values can be used by any command that uses global scalar or vector values from a compute as input. See this section for an overview of LIGGGHTS(R)-WSU output options.

The scalar and vector values calculated by this compute are "intensive". The scalar and vector values will be in pressure units.

Restrictions: none

Related commands:

compute temp, compute stress/atom, thermo_style,

Default: none

compute property/atom command

Syntax:

```
compute ID group-ID property/atom input1 input2 ...
```

- ID, group-ID are documented in compute command
- property/atom = style name of this compute command
- input = one or more atom attributes

```
possible attributes = id, mol, type, mass,
                      x, y, z, xs, ys, zs, xu, yu, zu, ix, iy, iz,
                      vx, vy, vz, fx, fy, fz,
                      q, mux, muy, muz, mu,
                      radius, diameter, omegax, omegay, omegaz,
                      angmomx, angmomy, angmomz,
                      shapex,shapey, shapez,
                      quatw, quati, quatj, quatk, tqx, tqy, tqz,
                      end1x, end1y, end1z, end2x, end2y, end2z,
                      corner1x, corner1y, corner1z,
                      corner2x, corner2y, corner2z,
                      corner3x, corner3y, corner3z,
                      i_name, d_name
```

```
        id = atom ID
        mol = molecule ID
        type = atom type
        mass = atom mass
        x,y,z = unscaled atom coordinates
        xs,ys,zs = scaled atom coordinates
        xu,yu,zu = unwrapped atom coordinates
        ix,iy,iz = box image that the atom is in
        vx,vy,vz = atom velocities
        fx,fy,fz = forces on atoms
        q = atom charge
        mux,muy,muz = orientation of dipole moment of atom
        mu = magnitude of dipole moment of atom
        radius,diameter = radius,diameter of spherical particle
        omegax,omegay,omegaz = angular velocity of spherical particle
        angmomx,angmomy,angmomz = angular momentum of aspherical particle
        shapex,shapey,shapez = 3 diameters of aspherical particle
        quatw,quati,quatj,quatk = quaternion components for aspherical or body particles
        tqx,tqy,tqz = torque on finite-size particles
        end12x, end12y, end12z = end points of line segment
        coner123x, corner123y, corner123z = corner points of triangle
        i_name = custom integer vector with name
        d_name = custom integer vector with name
```

Examples:

```
compute 1 all property/atom xs vx fx mux
compute 2 all property/atom type
compute 1 all property/atom ix iy iz
```

Description:

Define a computation that simply stores atom attributes for each atom in the group. This is useful so that the values can be used by other output commands that take computes as inputs. See for example, the compute

<u>reduce</u>, <u>fix ave/atom</u>, <u>fix ave/histo</u>, <u>fix ave/spatial</u>, and <u>atom-style variable</u> commands.

The list of possible attributes is the same as that used by the <u>dump custom</u> command, which describes their meaning, with some additional quantities that are only defined for certain <u>atom styles</u>. Basically, this list gives your input script access to any per-atom quantity stored by LIGGGHTS(R)-WSU.

The values are stored in a per-atom vector or array as discussed below. Zeroes are stored for atoms not in the specified group or for quantities that are not defined for a particular particle in the group (e.g. *shapex* if the particle is not an ellipsoid).

The additional quantities only accessible via this command, and not directly via the <u>dump custom</u> command, are as follows.

Shapex, *shapey*, and *shapez* are defined for ellipsoidal particles and define the 3d shape of each particle.

Quatw, *quati*, *quatj*, and *quatk* are defined for ellipsoidal particles and body particles and store the 4-vector quaternion representing the orientation of each particle. See the <u>set</u> command for an explanation of the quaternion vector.

End1x, *end1y*, *end1z*, *end2x*, *end2y*, *end2z*, are defined for line segment particles and define the end points of each line segment.

Corner1x, *corner1y*, *corner1z*, *corner2x*, *corner2y*, *corner2z*, *corner3x*, *corner3y*, *corner3z*, are defined for triangular particles and define the corner points of each triangle.

The *i_name* and *d_name* attributes refer to custom integer and floating-point properties that have been added to each atom via the <u>fix property/atom</u> command. When that command is used specific names are given to each attribute which are what is specified as the "name" portion of *i_name* or *d_name*.

Output info:

This compute calculates a per-atom vector or per-atom array depending on the number of input values. If a single input is specified, a per-atom vector is produced. If two or more inputs are specified, a per-atom array is produced where the number of columns = the number of inputs. The vector or array can be accessed by any command that uses per-atom values from a compute as input. See <u>this section</u> for an overview of LIGGGHTS(R)-WSU output options.

The vector or array values will be in whatever <u>units</u> the corresponding attribute is in, e.g. velocity units for vx, charge units for q, etc.

Restrictions: none

Related commands:

<u>dump custom</u>, <u>compute reduce</u>, <u>fix ave/atom</u>, <u>fix ave/spatial</u>, <u>fix property/atom</u>

Default: none

compute property/local command

Syntax:

```
compute ID group-ID property/local input1 input2 ...
```

- ID, group-ID are documented in [compute](#) command
- property/local = style name of this compute command
- input = one or more attributes

```
        possible attributes = natom1 natom2 ntype1 ntype2
                              patom1 patom2 ptype1 ptype2
                              batom1 batom2 btype

        natom1, natom2 = IDs of 2 atoms in each pair (within neighbor cutoff)
        ntype1, ntype2 = type of 2 atoms in each pair (within neighbor cutoff)
        patom1, patom2 = IDs of 2 atoms in each pair (within force cutoff)
        ptype1, ptype2 = type of 2 atoms in each pair (within force cutoff)
        batom1, batom2 = IDs of 2 atoms in each bond
        btype = bond type of each bond
```

Examples:

```
compute 1 all property/local btype batom1 batom2
compute 1 all property/local atype aatom2
```

Description:

Define a computation that stores the specified attributes as local data so it can be accessed by other [output commands](#). If the input attributes refer to bond information, then the number of datums generated, aggregated across all processors, equals the number of bonds in the system. Ditto for pairs.

If multiple input attributes are specified then they must all generate the same amount of information, so that the resulting local array has the same number of rows for each column. This means that only bond attributes can be specified together.

If the inputs are pair attributes, the local data is generated by looping over the pairwise neighbor list. Info about an individual pairwise interaction will only be included if both atoms in the pair are in the specified compute group. For *natom1* and *natom2*, all atom pairs in the neighbor list are considered (out to the neighbor cutoff = force cutoff + [neighbor skin](#)). For *patom1* and *patom2*, the distance between the atoms must be less than the force cutoff distance for that pair to be included, as defined by the [pair_style](#) and [pair_coeff](#) commands.

If the inputs are bond, etc attributes, the local data is generated by looping over all the atoms owned on a processor and extracting bond, etc info. For bonds, info about an individual bond will only be included if both atoms in the bond are in the specified compute group. Likewise for angles, dihedrals, etc.

Note that as atoms migrate from processor to processor, there will be no consistent ordering of the entries within the local vector or array from one timestep to the next. The only consistency that is guaranteed is that the ordering on a particular timestep will be the same for local vectors or arrays generated by other compute commands. For example, output from the [compute bond/local](#) command can be combined with bond atom indices from this command and output by the [dump local](#) command in a consistent way.

The *natom1* and *natom2*, or *patom1* and *patom2* attributes refer to the atom IDs of the 2 atoms in each pairwise interaction computed by the pair_style command. The *ntype1* and *ntype2*, or *ptype1* and *ptype2* attributes refer to the atom types of the 2 atoms in each pairwise interaction.

IMPORTANT NOTE: For pairs, if two atoms I,J are involved in 1-2, 1-3, 1-4 interactions within the molecular topology, their pairwise interaction may be turned off, and thus they may not appear in the neighbor list, and will not be part of the local data created by this command. More specifically, this may be true of I,J pairs with a weighting factor of 0.0; pairs with a non-zero weighting factor are included. The weighting factors for 1-2, 1-3, and 1-4 pairwise interactions are set by the special_bonds command.

The *batom1* and *batom2* attributes refer to the atom IDs of the 2 atoms in each bond. The *btype* attribute refers to the type of the bond, from 1 to Nbtypes = # of bond types. The number of bond types is defined in the data file read by the read_data command.

Output info:

This compute calculates a local vector or local array depending on the number of input values. The length of the vector or number of rows in the array is the number of bonds. If a single input is specified, a local vector is produced. If two or more inputs are specified, a local array is produced where the number of columns = the number of inputs. The vector or array can be accessed by any command that uses local values from a compute as input. See this section for an overview of LIGGGHTS(R)-WSU output options.

The vector or array values will be integers that correspond to the specified attribute.

Restrictions: none

Related commands:

dump local, compute reduce

Default: none

compute property/molecule command

Syntax:

```
compute ID group-ID property/molecule input1 input2 ...
```

- ID, group-ID are documented in compute command
- property/molecule = style name of this compute command
- input = one or more attributes

```
    possible attributes = mol cout
      mol = molecule ID
      count = # of atoms in molecule
```

Examples:

```
compute 1 all property/molecule mol
```

Description:

Define a computation that stores the specified attributes as global data so it can be accessed by other output commands and used in conjunction with other commands that generate per-molecule data, such as compute com/molecule and compute msd/molecule.

The ordering of per-molecule quantities produced by this compute is consistent with the ordering produced by other compute commands that generate per-molecule datums. Conceptually, them molecule IDs will be in ascending order for any molecule with one or more of its atoms in the specified group.

The *mol* attribute is the molecule ID. This attribute can be used to produce molecule IDs as labels for per-molecule datums generated by other computes or fixes when they are output to a file, e.g. by the fix ave/time command.

The *count* attribute is the number of atoms in the molecule.

Output info:

This compute calculates a global vector or global array depending on the number of input values. The length of the vector or number of rows in the array is the number of molecules. If a single input is specified, a global vector is produced. If two or more inputs are specified, a global array is produced where the number of columns = the number of inputs. The vector or array can be accessed by any command that uses global values from a compute as input. See this section for an overview of LIGGGHTS(R)-WSU output options.

The vector or array values will be integers that correspond to the specified attribute.

Restrictions: none

Related commands: none

Default: none

compute rdf command

Syntax:

```
compute ID group-ID rdf Nbin itype1 jtype1 itype2 jtype2 ...
```

- ID, group-ID are documented in **compute** command
- rdf = style name of this compute command
- Nbin = number of RDF bins
- itypeN = central atom type for Nth RDF histogram (see asterisk form below)
- jtypeN = distribution atom type for Nth RDF histogram (see asterisk form below)

Examples:

```
compute 1 all rdf 100
compute 1 all rdf 100 1 1
compute 1 all rdf 100 * 3
compute 1 fluid rdf 500 1 1 1 2 2 1 2 2
compute 1 fluid rdf 500 1*3 2 5 *10
```

Description:

Define a computation that calculates the radial distribution function (RDF), also called g(r), and the coordination number for a group of particles. Both are calculated in histogram form by binning pairwise distances into *Nbin* bins from 0.0 to the maximum force cutoff defined by the pair_style command. The bins are of uniform size in radial distance. Thus a single bin encompasses a thin shell of distances in 3d and a thin ring of distances in 2d.

IMPORTANT NOTE: If you have a bonded system, then the settings of special_bonds command can remove pairwise interactions between atoms in the same bond, angle, or dihedral. This is the default setting for the special_bonds command, and means those pairwise interactions do not appear in the neighbor list. Because this fix uses the neighbor list, it also means those pairs will not be included in the RDF. One way to get around this, is to write a dump file, and use the rerun command to compute the RDF for snapshots in the dump file. The rerun script can use a special_bonds command that includes all pairs in the neighbor list.

The *itypeN* and *jtypeN* arguments are optional. These arguments must come in pairs. If no pairs are listed, then a single histogram is computed for g(r) between all atom types. If one or more pairs are listed, then a separate histogram is generated for each *itype,jtype* pair.

The *itypeN* and *jtypeN* settings can be specified in one of two ways. An explicit numeric value can be used, as in the 4th example above. Or a wild-card asterisk can be used to specify a range of atom types. This takes the form "*" or "*n" or "n*" or "m*n". If N = the number of atom types, then an asterisk with no numeric values means all types from 1 to N. A leading asterisk means all types from 1 to n (inclusive). A trailing asterisk means all types from n to N (inclusive). A middle asterisk means all types from m to n (inclusive).

If both *itypeN* and *jtypeN* are single values, as in the 4th example above, this means that a g(r) is computed where atoms of type *itypeN* are the central atom, and atoms of type *jtypeN* are the distribution atom. If either *itypeN* and *jtypeN* represent a range of values via the wild-card asterisk, as in the 5th example above, this means that a g(r) is computed where atoms of any of the range of types represented by *itypeN* are the central atom, and atoms of any of the range of types represented by *jtypeN* are the distribution atom.

Pairwise distances are generated by looping over a pairwise neighbor list, just as they would be in a [pair_style](#) computation. The distance between two atoms I and J is included in a specific histogram if the following criteria are met:

- atoms I,J are both in the specified compute group
- the distance between atoms I,J is less than the maximum force cutoff
- the type of the I atom matches itypeN (one or a range of types)
- the type of the J atom matches jtypeN (one or a range of types)

It is OK if a particular pairwise distance is included in more than one individual histogram, due to the way the *itypeN* and *jtypeN* arguments are specified.

The g(r) value for a bin is calculated from the histogram count by scaling it by the idealized number of how many counts there would be if atoms of type *jtypeN* were uniformly distributed. Thus it involves the count of *itypeN* atoms, the count of *jtypeN* atoms, the volume of the entire simulation box, and the volume of the bin's thin shell in 3d (or the area of the bin's thin ring in 2d).

A coordination number coord(r) is also calculated, which is the number of atoms of type *jtypeN* within the current bin or closer, averaged over atoms of type *itypeN*. This is calculated as the area- or volume-weighted sum of g(r) values over all bins up to and including the current bin, multiplied by the global average volume density of atoms of type jtypeN.

The simplest way to output the results of the compute rdf calculation to a file is to use the [fix ave/time](#) command, for example:

```
compute myRDF all rdf 50
fix 1 all ave/time 100 1 100 c_myRDF file tmp.rdf mode vector
```

Output info:

This compute calculates a global array with the number of rows = *Nbins*, and the number of columns = 1 + 2*Npairs, where Npairs is the number of I,J pairings specified. The first column has the bin coordinate (center of the bin), Each successive set of 2 columns has the g(r) and coord(r) values for a specific set of *itypeN* versus *jtypeN* interactions, as described above. These values can be used by any command that uses a global values from a compute as input. See [Section_howto 15](#) for an overview of LIGGGHTS(R)-WSU output options.

The array values calculated by this compute are all "intensive".

The first column of array values will be in distance [units](#). The g(r) columns of array values are normalized numbers >= 0.0. The coordination number columns of array values are also numbers >= 0.0.

Restrictions:

The RDF is not computed for distances longer than the force cutoff, since processors (in parallel) don't know about atom coordinates for atoms further away than that distance. If you want an RDF for larger distances, you can use the [rerun](#) command to post-process a dump file. The definition of g(r) used by LIGGGHTS(R)-WSU is only appropriate for characterizing atoms that are uniformly distributed throughout the simulation cell. In such cases, the coordination number is still correct and meaningful. As an example, if a large simulation cell contains only one atom of type *itypeN* and one of *jtypeN*, then g(r) will register an arbitrarily large spike at whatever distance they happen to be at, and zero everywhere else. coord(r) will show a step change from zero to one at the location of the spike in g(r).

Related commands:

fix ave/time

Default: none

fix ave/time

Default: none

compute reduce command

compute reduce/region command

Syntax:

```
compute ID group-ID style arg mode input1 input2 ... keyword args ...
```

- ID, group-ID are documented in compute command
- style = *reduce* or *reduce/region*

```
  reduce arg = none
    reduce/region arg = region-ID
      region-ID = ID of region to use for choosing atoms
```
- mode = *sum* or *min* or *max* or *ave*
- one or more inputs can be listed
- input = x, y, z, vx, vy, vz, fx, fy, fz, c_ID, c_ID[N], f_ID, f_ID[N], v_name

```
  x,y,z,vx,vy,vz,fx,fy,fz = atom attribute (position, velocity, force component)
  c_ID = per-atom or local vector calculated by a compute with ID
  c_ID[I] = Ith column of per-atom or local array calculated by a compute with ID
  f_ID = per-atom or local vector calculated by a fix with ID
  f_ID[I] = Ith column of per-atom or local array calculated by a fix with ID
  v_name = per-atom vector calculated by an atom-style variable with name
```
- zero or more keyword/args pairs may be appended
- keyword = *replace*

```
  replace args = vec1 vec2
      vec1 = reduced value from this input vector will be replaced
      vec2 = replace it with vec1[N] where N is index of max/min value from vec2
```

Examples:

```
compute 1 all reduce sum c_force
compute 1 all reduce/region subbox sum c_force
compute 2 all reduce min c_press2 f_ave v_myKE
compute 3 fluid reduce max c_index1 c_index2 c_dist replace 1 3 replace 2 3
```

Description:

Define a calculation that "reduces" one or more vector inputs into scalar values, one per listed input. The inputs can be per-atom or local quantities; they cannot be global quantities. Atom attributes are per-atom quantities, computes and fixes may generate any of the three kinds of quantities, and atom-style variables generate per-atom quantities. See the variable command and its special functions which can perform the same operations as the compute reduce command on global vectors.

The reduction operation is specified by the *mode* setting. The *sum* option adds the values in the vector into a global total. The *min* or *max* options find the minimum or maximum value across all vector values. The *ave* setting adds the vector values into a global total, then divides by the number of values in the vector.

Each listed input is operated on independently. For per-atom inputs, the group specified with this command means only atoms within the group contribute to the result. For per-atom inputs, if the compute reduce/region command is used, the atoms must also currently be within the region. Note that an input that produces per-atom quantities may define its own group which affects the quantities it returns. For example, if a

compute is used as an input which generates a per-atom vector, it will generate values of 0.0 for atoms that are not in the group specified for that compute.

Each listed input can be an atom attribute (position, velocity, force component) or can be the result of a compute or fix or the evaluation of an atom-style variable.

The atom attribute values (x,y,z,vx,vy,vz,fx,fy,fz) are self-explanatory. Note that other atom attributes can be used as inputs to this fix by using the compute property/atom command and then specifying an input value from that compute.

If a value begins with "c_", a compute ID must follow which has been previously defined in the input script. Computes can generate per-atom or local quantities. See the individual compute doc page for details. If no bracketed integer is appended, the vector calculated by the compute is used. If a bracketed integer is appended, the Ith column of the array calculated by the compute is used. Users can also write code for their own compute styles and add them to LIGGGHTS(R)-WSU.

If a value begins with "f_", a fix ID must follow which has been previously defined in the input script. Fixes can generate per-atom or local quantities. See the individual fix doc page for details. Note that some fixes only produce their values on certain timesteps, which must be compatible with when compute reduce references the values, else an error results. If no bracketed integer is appended, the vector calculated by the fix is used. If a bracketed integer is appended, the Ith column of the array calculated by the fix is used. Users can also write code for their own fix style and add them to LIGGGHTS(R)-WSU.

If a value begins with "v_", a variable name must follow which has been previously defined in the input script. It must be an atom-style variable. Atom-style variables can reference thermodynamic keywords and various per-atom attributes, or invoke other computes, fixes, or variables when they are evaluated, so this is a very general means of generating per-atom quantities to reduce.

If the *replace* keyword is used, two indices *vec1* and *vec2* are specified, where each index ranges from 1 to the # of input values. The replace keyword can only be used if the *mode* is *min* or *max*. It works as follows. A min/max is computed as usual on the *vec2* input vector. The index N of that value within *vec2* is also stored. Then, instead of performing a min/max on the *vec1* input vector, the stored index is used to select the Nth element of the *vec1* vector.

Thus, for example, if you wish to use this compute to find the bond with maximum stretch, you can do it as follows:

```
compute 1 all property/local batom1 batom2
compute 2 all bond/local dist
compute 3 all reduce max c_1[1] c_1[2] c_2 replace 1 3 replace 2 3
thermo_style custom step temp c_3[1] c_3[2] c_3[3]
```

The first two input values in the compute reduce command are vectors with the IDs of the 2 atoms in each bond, using the compute property/local command. The last input value is bond distance, using the compute bond/local command. Instead of taking the max of the two atom ID vectors, which does not yield useful information in this context, the *replace* keywords will extract the atom IDs for the two atoms in the bond of maximum stretch. These atom IDs and the bond stretch will be printed with thermodynamic output.

If a single input is specified this compute produces a global scalar value. If multiple inputs are specified, this compute produces a global vector of values, the length of which is equal to the number of inputs specified.

As discussed below, for *sum* mode, the value(s) produced by this compute are all "extensive", meaning their value scales linearly with the number of atoms involved. If normalized values are desired, this compute can be accessed by the thermo_style custom command with thermo_modify norm yes set as an option. Or it can be accessed by a variable that divides by the appropriate atom count.

Output info:

This compute calculates a global scalar if a single input value is specified or a global vector of length N where N is the number of inputs, and which can be accessed by indices 1 to N. These values can be used by any command that uses global scalar or vector values from a compute as input. See Section_howto 15 for an overview of LIGGGHTS(R)-WSU output options.

All the scalar or vector values calculated by this compute are "intensive", except when the *sum* mode is used on per-atom or local vectors, in which case the calculated values are "extensive".

The scalar or vector values will be in whatever units the quantities being reduced are in.

Restrictions: none

Related commands:

compute, fix, variable

Default: none

compute rigid command

compute multisphere command

Syntax:

```
compute ID group-ID rigid (or multisphere) property property_name
```

- ID, group-ID are documented in compute command
- property = obligatory keyword
- property_name = *xcm* or *vcm* or *fcm* or *torque* or *quat* or *angmom* or *omega* or *density* or *type* or *id* or *masstotal* or *inertia* or *ex_space* or *ey_space* or *ez_space*

```
xcm = body position (based on center of mass) (3 values)
  vcm = body velocity (based on center of mass) (3 values)
  fcm = body force (based on center of mass) (3 values)
  torque = body torque (based on center of mass) (3 values)
  quat = body quaternion (based on center of mass) (4 values)
  angmom = body angular momentum (based on center of mass) (3 values)
  omega = body angular velocity (based on center of mass) (3 values)
  density = body density (1 value)
  atomtype = atom type (material type) of the rigid body (1 value)
  clumptype = multi-sphere type as defined in fix paticleteplate/multisphere (1 value)
  id_multisphere = body id (1 value)
  masstotal = body mass (1 value)
  inertia = body intertia (based on center of mass, around ex_space, ey_space, ez_space) (
  ex_space, ey_space, ez_space = eigensystem of the body (based on center of mass) (3 value
```

Examples:

```
compute xcm all rigid property xcm
compute xcm all multisphere property xcm
```

Description:

Define a computation that calculates properties of individual multi-sphere bodies (clumps) in the similation that were defined via fix particletemplate/multisphere

The local data stored by this command is generated by looping over the all the bodies owned on a process.

IMPORTANT NOTE: the group-ID is ignored for this command, as group data is atom-based, not clump-based.

Output info:

This compute calculates a local vector or local array depending on the length of the data (see above). The vector or array can be accessed by any command that uses local values from a compute as input. See this section for an overview of LIGGGHTS(R)-WSU output options.

Restrictions:

Can only be used together with a granular pair style. For accessing particle-wall contact data, only mesh walls can be used.

Related commands:

[dump local](#), [compute property/local](#)

Default: none

compute slice command

Syntax:

```
compute ID group-ID slice Nstart Nstop Nskip input1 input2 ...
```

- ID, group-ID are documented in [compute](#) command
- slice = style name of this compute command
- Nstart = starting index within input vector(s)
- Nstop = stopping index within input vector(s)
- Nskip = extract every Nskip elements from input vector(s)
- input = c_ID, c_ID[N], f_ID, f_ID[N]

```
c_ID = global vector calculated by a compute with ID
c_ID[I] = Ith column of global array calculated by a compute with ID
f_ID = global vector calculated by a fix with ID
f_ID[I] = Ith column of global array calculated by a fix with ID
```

Examples:

```
compute 1 all slice 1 100 10 c_msdmol[4]
compute 1 all slice 301 400 1 c_msdmol[4]
```

Description:

Define a calculation that "slices" one or more vector inputs into smaller vectors, one per listed input. The inputs can be global quantities; they cannot be per-atom or local quantities. [Computes](#) and [fixes](#) may generate any of the three kinds of quantities. [Variables](#) do not generate global vectors. The group specified with this command is ignored.

The values extracted from the input vector(s) are determined by the *Nstart*, *Nstop*, and *Nskip* parameters. The elements of an input vector of length N are indexed from 1 to N. Starting at element *Nstart*, every Mth element is extracted, where M = *Nskip*, until element *Nstop* is reached. The extracted quantities are stored as a vector, which is typically shorter than the input vector.

Each listed input is operated on independently to produce one output vector. Each listed input must be a global vector or column of a global array calculated by another [compute](#) or [fix](#).

If an input value begins with "c_", a compute ID must follow which has been previously defined in the input script and which generates a global vector or array. See the individual [compute](#) doc page for details. If no bracketed integer is appended, the vector calculated by the compute is used. If a bracketed integer is appended, the Ith column of the array calculated by the compute is used. Users can also write code for their own compute styles and [add them to LIGGGHTS(R)-WSU](#).

If a value begins with "f_", a fix ID must follow which has been previously defined in the input script and which generates a global vector or array. See the individual [fix](#) doc page for details. Note that some fixes only produce their values on certain timesteps, which must be compatible with when compute slice references the values, else an error results. If no bracketed integer is appended, the vector calculated by the fix is used. If a bracketed integer is appended, the Ith column of the array calculated by the fix is used. Users can also write code for their own fix style and [add them to LIGGGHTS(R)-WSU](#).

If a single input is specified this compute produces a global vector, even if the length of the vector is 1. If

multiple inputs are specified, then a global array of values is produced, with the number of columns equal to the number of inputs specified.

Output info:

This compute calculates a global vector if a single input value is specified or a global array with N columns where N is the number of inputs. The length of the vector or the number of rows in the array is equal to the number of values extracted from each input vector. These values can be used by any command that uses global vector or array values from a compute as input. See [this section](#) for an overview of LIGGGHTS(R)-WSU output options.

The vector or array values calculated by this compute are simply copies of values generated by computes or fixes that are input vectors to this compute. If there is a single input vector of intensive and/or extensive values, then each value in the vector of values calculated by this compute will be "intensive" or "extensive", depending on the corresponding input value. If there are multiple input vectors, and all the values in them are intensive, then the array values calculated by this compute are "intensive". If there are multiple input vectors, and any value in them is extensive, then the array values calculated by this compute are "extensive".

The vector or array values will be in whatever [units](#) the input quantities are in.

Restrictions: none

Related commands:

[compute](#), [fix](#), [compute reduce](#)

Default: none

compute stress/atom command

Syntax:

```
compute ID group-ID stress/atom keyword ...
```

- ID, group-ID are documented in [compute](#) command
- stress/atom = style name of this compute command
- zero or more keywords may be appended
- keyword = *ke* or *pair* or *bond* or *angle* or *dihedral* or *improper* or *kspace* or *fix* or *virial*

Examples:

```
compute 1 mobile stress/atom
compute 1 all stress/atom pair bond
```

Description:

Define a computation that computes the symmetric per-atom stress tensor for each atom in a group. The tensor for each atom has 6 components and is stored as a 6-element vector in the following order: xx, yy, zz, xy, xz, yz. See the [compute pressure](#) command if you want the stress tensor (pressure) of the entire system.

The stress tensor for atom *I* is given by the following formula, where *a* and *b* take on values x,y,z to generate the 6 components of the symmetric tensor:

$$
\begin{aligned}
S_{ab} = -\Bigg[& mv_a v_b + \frac{1}{2}\sum_{n=1}^{N_p}(r_{1_a}F_{1_b} + r_{2_a}F_{2_b}) + \frac{1}{2}\sum_{n=1}^{N_b}(r_{1_a}F_{1_b} + r_{2_a}F_{2_b}) + \\
& \frac{1}{3}\sum_{n=1}^{N_a}(r_{1_a}F_{1_b} + r_{2_a}F_{2_b} + r_{3_a}F_{3_b}) + \frac{1}{4}\sum_{n=1}^{N_d}(r_{1_a}F_{1_b} + r_{2_a}F_{2_b} + r_{3_a}F_{3_b} + r_{4_a}F_{4_b}) + \\
& \frac{1}{4}\sum_{n=1}^{N_i}(r_{1_a}F_{1_b} + r_{2_a}F_{2_b} + r_{3_a}F_{3_b} + r_{4_a}F_{4_b}) + \text{Kspace}(r_{i_a}, F_{i_b}) + \sum_{n=1}^{N_f} r_{i_a}F_{i_b} \Bigg]
\end{aligned}
$$

The first term is a kinetic energy contribution for atom *I*. The second term is a pairwise energy contribution where *n* loops over the *Np* neighbors of atom *I*, *r1* and *r2* are the positions of the 2 atoms in the pairwise interaction, and *F1* and *F2* are the forces on the 2 atoms resulting from the pairwise interaction. The third term is a bond contribution of similar form for the *Nb* bonds which atom *I* is part of. There are similar terms for the *Na* angle, *Nd* dihedral, and *Ni* improper interactions atom *I* is part of. There is also a term for the KSpace contribution from long-range Coulombic interactions, if defined. Finally, there is a term for the *Nf* [fixes](#) that apply internal constraint forces to atom *I*. Currently, only the [fix shake](#) and [fix rigid](#) commands contribute to this term.

IMPORTANT NOTE: For granular systems, this formular neglects the contribution of average velocity in the kinetic energy contribution. This is corrected in the compute ave/euler command (currently no doc available).

As the coefficients in the formula imply, a virial contribution produced by a small set of atoms (e.g. 4 atoms in a dihedral or 3 atoms in a Tersoff 3-body interaction) is assigned in equal portions to each atom in the set. E.g. 1/4 of the dihedral virial to each of the 4 atoms, or 1/3 of the fix virial due to SHAKE constraints applied to atoms in a a water molecule via the fix shake command.

If no extra keywords are listed, all of the terms in this formula are included in the per-atom stress tensor. If any extra keywords are listed, only those terms are summed to compute the tensor. The *virial* keyword means include all terms except the kinetic energy *ke*.

Note that the stress for each atom is due to its interaction with all other atoms in the simulation, not just with other atoms in the group.

The dihedral_style charmm style calculates pairwise interactions between 1-4 atoms. The virial contribution of these terms is included in the pair virial, not the dihedral virial.

The KSpace contribution is calculated using the method in (Heyes) for the Ewald method and by the methodology described in (Sirk) for PPPM. The choice of KSpace solver is specified by the kspace_style pppm command. Note that for PPPM, the calcluation requires 6 extra FFTs each timestep that per-atom stress is calculated. Thus it can significantly increase the cost of the PPPM calculation if it is needed on a large fraction of the simulation timesteps.

Note that as defined in the formula, per-atom stress is the negative of the per-atom pressure tensor. It is also really a stress*volume formulation, meaning the computed quantity is in units of pressure*volume. It would need to be divided by a per-atom volume to have units of stress (pressure), but an individual atom's volume is not well defined or easy to compute in a deformed solid or a liquid. Thus, if the diagonal components of the per-atom stress tensor are summed for all atoms in the system and the sum is divided by dV, where d = dimension and V is the volume of the system, the result should be -P, where P is the total pressure of the system.

These lines in an input script for a 3d system should yield that result. I.e. the last 2 columns of thermo output will be the same:

```
compute         peratom all stress/atom
compute         p all reduce sum c_peratom[1] c_peratom[2] c_peratom[3]
variable        press equal -(c_p[1]+c_p[2]+c_p[3])/(3*vol)
thermo_style    custom step temp etotal press v_press
```

Output info:

This compute calculates a per-atom array with 6 columns, which can be accessed by indices 1-6 by any command that uses per-atom values from a compute as input. See Section_howto 15 for an overview of LIGGGHTS(R)-WSU output options.

The per-atom array values will be in pressure*volume units as discussed above.

Restrictions: none

Related commands:

compute pe, compute pressure

Default: none

(Heyes) Heyes, Phys Rev B 49, 755 (1994),

(Sirk) Sirk, Moore, Brown, J Chem Phys, 138, 064505 (2013).

compute voronoi/atom command

Syntax:

```
compute ID group-ID voronoi/atom keyword arg ...
```

- ID, group-ID are documented in [compute](#) command
- voronoi/atom = style name of this compute command
- zero or more keyword/value pairs may be appended
- keyword = *only_group* or *surface* or *radius* or *edge_histo* or *edge_threshold* or *face_threshold*

```
  only_group = no arg
    surface arg = sgroup-ID
      sgroup-ID = compute the dividing surface between group-ID and sgroup-ID
        this keyword adds a third column to the compute output
    radius arg = v_r
      v_r = radius atom style variable for a poly-disperse voronoi tessellation
    edge_histo arg = maxedge
      maxedge = maximum number of voronoi cell edges to be accounted in the histogram
    edge_threshold arg = minlength
      minlength = minimum length for an edge to be counted
    face_threshold arg = minarea
      minarea = minimum area for a face to be counted
```

Examples:

```
compute 1 all voronoi/atom
compute 2 precipitate voronoi/atom surface matrix
compute 3b precipitate voronoi/atom radius v_r
compute 4 solute voronoi/atom only_group
```

Description:

Define a computation that calculates the Voronoi tessellation of the atoms in the simulation box. The tessellation is calculated using all atoms in the simulation, but non-zero values are only stored for atoms in the group.

By default two quantities per atom are calculated by this compute. The first is the volume of the Voronoi cell around each atom. Any point in an atom's Voronoi cell is closer to that atom than any other. The second is the number of faces of the Voronoi cell, which is also the number of nearest neighbors of the atom in the middle of the cell.

If the *only_group* keyword is specified the tessellation is performed only with respect to the atoms contained in the compute group. This is equivalent to deleting all atoms not contained in the group prior to evaluating the tessellation.

If the *surface* keyword is specified a third quantity per atom is computed: the voronoi cell surface of the given atom. *surface* takes a group ID as an argument. If a group other than *all* is specified, only the voronoi cell facets facing a neighbor atom from the specified group are counted towards the surface area.

In the example above, a precipitate embedded in a matrix, only atoms at the surface of the precipitate will have non-zero surface area, and only the outward facing facets of the voronoi cells are counted (the hull of the precipitate). The total surface area of the precipitate can be obtained by running a "reduce sum" compute on c_2[3]

If the *radius* keyword is specified with an atom style variable as the argument, a poly-disperse voronoi tessellation is performed. Examples for radius variables are

```
variable r1 atom (type==1)*0.1+(type==2)*0.4
compute radius all property/atom radius
variable r2 atom c_radius
```

Here v_r1 specifies a per-type radius of 0.1 units for type 1 atoms and 0.4 units for type 2 atoms, and v_r2 accesses the radius property present in atom_style sphere for granular models.

The *edge_histo* keyword activates the compilation of a histogram of number of edges on the faces of the voronoi cells in the compute group. The argument maxedge of the this keyword is the largest number of edges on a single voronoi cell face expected to occur in the sample. This keyword adds the generation of a global vector with maxedge+1 entries. The last entry in the vector contains the number of faces with with more than maxedge edges. Since the polygon with the smallest amount of edges is a triangle, entries 1 and 2 of the vector will always be zero.

The *edge_threshold* and *face_threshold* keywords allow the suppression of edges below a given minimum length and faces below a given minimum area. Ultra short edges and ultra small faces can occur as artifacts of the voronoi tessellation. These keywords will affect the neighbor count and edge histogram outputs.

The Voronoi calculation is performed by the freely available Voro++ package, written by Chris Rycroft at UC Berkeley and LBL, which must be installed on your system when building LIGGGHTS(R)-WSU for use with this compute. See instructions on obtaining and installing the Voro++ software in the src/VORONOI/README file.

IMPORTANT NOTE: The calculation of Voronoi volumes is performed by each processor for the atoms it owns, and includes the effect of ghost atoms stored by the processor. This assumes that the Voronoi cells of owned atoms are not affected by atoms beyond the ghost atom cut-off distance. This is usually a good assumption for liquid and solid systems, but may lead to underestimation of Voronoi volumes in low density systems. By default, the set of ghost atoms stored by each processor is determined by the cutoff used for pair_style interactions. The cutoff can be set explicitly via the communicate cutoff command.

IMPORTANT NOTE: The Voro++ package performs its calculation in 3d. This should still work for a 2d LIGGGHTS(R)-WSU simulation, to effectively compute Voronoi "areas", so long as the z-dimension of the box is roughly the same (or smaller) compared to the separation of the atoms. Typical values for the z box dimensions in a 2d LIGGGHTS(R)-WSU model are -0.5 to 0.5, which satisfies the criterion for most units systems. Note that you define the z extent of the simulation box for 2d simulations when using the create_box or read_data commands.

Output info:

This compute calculates a per-atom array with 2 columns. The first column is the Voronoi volume, the second is the neighbor count, as described above. These values can be accessed by any command that uses per-atom values from a compute as input. See Section_howto 15 for an overview of LIGGGHTS(R)-WSU output options.

The Voronoi cell volume will be in distance units cubed.

Restrictions:

This compute is part of the VORONOI package. It is only enabled if LIGGGHTS(R)-WSU was built with that package. See the Making LIGGGHTS(R)-WSU section for more info.

Related commands:

compute voronoi/atom command 95

dump custom

Default: none

create_atoms command

Syntax:

```
create_atoms type style args keyword values ...
```

- type = atom type (1-Ntypes) of atoms to create
- style = *box* or *region* or *single* or *random*

```
  box args = none
    region args = region-ID
      region-ID = atoms will only be created if contained in the region
    single args = x y z
      x,y,z = coordinates of a single atom (distance units)
    random args = N seed region-ID
      N = number of atoms to create
      seed = random # seed (positive integer)
      region-ID = create atoms within this region, use NULL for entire simulation box
```

- zero or more keyword/value pairs may be appended
- keyword = *basis* or *remap* or *units* or *all_in*

```
  basis values = M itype
      M = which basis atom
      itype = atom type (1-N) to assign to this basis atom
    remap value = yes or no
    units value = lattice or box
      lattice = the geometry is defined in lattice units
      box = the geometry is defined in simulation box units
    all_in value = all_in_dist
      all_in_dist = distance from region boundary for insertion
```

Examples:

```
create_atoms 1 box
create_atoms 3 region regsphere basis 2 3
create_atoms 3 single 0 0 5
```

Description:

This command creates atoms on a lattice, or a single atom, or a random collection of atoms, as an alternative to reading in their coordinates explicitly via a [read_data](#) or [read_restart](#) command. A simulation box must already exist, which is typically created via the [create_box](#) command. Before using this command, a lattice must also be defined using the [lattice](#) command. The only exceptions are for the *single* style with units = box or the *random* style.

For the *box* style, the create_atoms command fills the entire simulation box with atoms on the lattice. If your simulation box is periodic, you should insure its size is a multiple of the lattice spacings, to avoid unwanted atom overlaps at the box boundaries. If your box is periodic and a multiple of the lattice spacing in a particular dimension, LIGGGHTS(R)-WSU is careful to put exactly one atom at the boundary (on either side of the box), not zero or two.

For the *region* style, the geometric volume is filled that is inside the simulation box and is also consistent with the region volume. See the [region](#) command for details. Note that a region can be specified so that its "volume" is either inside or outside a geometric boundary. Also note that if your region is the same size as a periodic simulation box (in some dimension), LIGGGHTS(R)-WSU does not implement the same logic as

with the *box* style, to insure exactly one atom at the boundary. if this is what you desire, you should either use the *box* style, or tweak the region size to get precisely the atoms you want. With the optional *all_in* keyword, it can additionally be specified that the particles should be inserted a certain distance (as specified by *all_in_dist*) away from the region boundaries. *all_in* is only implemented for *region* insertion

For the *single* style, a single atom is added to the system at the specified coordinates. This can be useful for debugging purposes or to create a tiny system with a handful of atoms at specified positions.

For the *random* style, N atoms are added to the system at randomly generated coordinates, which can be useful for generating an amorphous system. The atoms are created one by one using the speficied random number *seed*, resulting in the same set of atom coordinates, independent of how many processors are being used in the simulation. If the *region-ID* argument is specified as NULL, then the created atoms will be anywhere in the simulation box. If a *region-ID* is specified, a geometric volume is filled that is inside the simulation box and is also consistent with the region volume. See the region command for details. Note that a region can be specified so that its "volume" is either inside or outside a geometric boundary.

IMPORTANT NOTE: The atoms generated by the *random* style will typically be highly overlapped which will cause many interatomic potentials to compute large energies and forces. Thus you should either perform an energy minimization or run dynamics with fix nve/limit to equilibrate such a system, before running normal dynamics.

The *basis* keyword specifies an atom type that will be assigned to specific basis atoms as they are created. See the lattice command for specifics on how basis atoms are defined for the unit cell of the lattice. By default, all created atoms are assigned the argument *type* as their atom type.

The *remap* keyword only applies to the *single* style. If it is set to *yes*, then if the specified position is outside the simulation box, it will mapped back into the box, assuming the relevant dimensions are periodic. If it is set to *no*, no remapping is done and no atom is created if its position is outside the box.

The *units* keyword determines the meaning of the distance units used to specify the coordinates of the one atom created by the *single* style. A *box* value selects standard distance units as defined by the units command, e.g. Angstroms for units = real or metal. A *lattice* value means the distance units are in lattice spacings.

Note that this command adds atoms to those that already exist. By using the create_atoms command multiple times, multiple sets of atoms can be added to the simulation. For example, interleaving create_atoms with lattice commands specifying different orientations, grain boundaries can be created. By using the create_atoms command in conjunction with the delete_atoms command, reasonably complex geometries can be created. The create_atoms command can also be used to add atoms to a system previously read in from a data or restart file. In all these cases, care should be taken to insure that new atoms do not overlap existing atoms inappropriately. The delete_atoms command can be used to handle overlaps.

Atom IDs are assigned to created atoms in the following way. The collection of created atoms are assigned consecutive IDs that start immediately following the largest atom ID existing before the create_atoms command was invoked. When a simulation is performed on different numbers of processors, there is no guarantee a particular created atom will be assigned the same ID.

Aside from their ID, atom type, and xyz position, other properties of created atoms are set to default values, depending on which quantities are defined by the chosen atom style. See the atom style command for more details. See the set and velocity commands for info on how to change these values.

- charge = 0.0
- diameter = 1.0
- shape = 0.0 0.0 0.0
- density = 1.0

- volume = 1.0
- velocity = 0.0 0.0 0.0
- angular velocity = 0.0 0.0 0.0
- angular momentum = 0.0 0.0 0.0
- quaternion = (1,0,0,0)
- bonds = none

Note that the *sphere* atom style sets the default particle diameter to 1.0 as well as the density. This means the mass for the particle is not 1.0, but is PI/6 * diameter^3 = 0.5236.

Note that the *ellipsoid* atom style sets the default particle shape to (0.0 0.0 0.0) and the density to 1.0 which means it is a point particle, not an ellipsoid, and has a mass of 1.0.

The set command can be used to override many of these default settings.

Restrictions:

An atom_style must be previously defined to use this command.

Related commands:

lattice, region, create_box, read_data, read_restart

Default:

The default for the *basis* keyword is that all created atoms are assigned the argument *type* as their atom type. The default for *remap* = no and for *units* = box.

create_box command

Syntax:

```
create_box N region-ID
```

- N = # of atom types to use in this simulation
- region-ID = ID of region to use as simulation domain

Examples:

```
create_box 2 mybox
```

Description:

This command creates a simulation box based on the specified region. Thus a region command must first be used to define a geometric domain.

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

If the region is not of style *prism*, then LIGGGHTS(R)-WSU encloses the region (block, sphere, etc) with an axis-aligned orthogonal bounding box which becomes the simulation domain.

If the region is of style *prism*, LIGGGHTS(R)-WSU creates a non-orthogonal simulation domain shaped as a parallelepiped with triclinic symmetry. As defined by the region prism command, the parallelepiped has its "origin" at (xlo,ylo,zlo) and is defined by 3 edge vectors starting from the origin given by A = (xhi-xlo,0,0); B = (xy,yhi-ylo,0); C = (xz,yz,zhi-zlo). *Xy,xz,yz* can be 0.0 or positive or negative values and are called "tilt factors" because they are the amount of displacement applied to faces of an originally orthogonal box to transform it into the parallelipiped.

A *prism* region used with the create_box command must have tilt factors (xy,xz,yz) that do not skew the box more than half the distance of the parallel box length. For example, if xlo = 2 and xhi = 12, then the x box length is 10 and the xy tilt factor must be between -5 and 5. Similarly, both xz and yz must be between -(xhi-xlo)/2 and +(yhi-ylo)/2. Note that this is not a limitation, since if the maximum tilt factor is 5 (as in this example), then configurations with tilt = ..., -15, -5, 5, 15, 25, ... are all geometrically equivalent.

See Section_howto 12 of the doc pages for a geometric description of triclinic boxes, as defined by LIGGGHTS(R)-WSU, and how to transform these parameters to and from other commonly used triclinic representations.

When a prism region is used, the simulation domain must be periodic in any dimensions with a non-zero tilt factor, as defined by the boundary command. I.e. if the xy tilt factor is non-zero, then both the x and y dimensions must be periodic. Similarly, x and z must be periodic if xz is non-zero and y and z must be periodic if yz is non-zero. Also note that if your simulation will tilt the box, e.g. via the fix deform command, the simulation box must be defined as triclinic, even if the tilt factors are initially 0.0.

IMPORTANT NOTE: If the system is non-periodic (in a dimension), then you should not make the lo/hi box dimensions (as defined in your region command) radically smaller/larger than the extent of the atoms you eventually plan to create, e.g. via the create_atoms command. For example, if your atoms extend from 0 to 50, you should not specify the box bounds as -10000 and 10000. This is because LIGGGHTS(R)-WSU uses the specified box size to layout the 3d grid of processors. A huge (mostly empty) box will be sub-optimal for

performance when using "fixed" boundary conditions (see the **boundary** command). When using "shrink-wrap" boundary conditions (see the **boundary** command), a huge (mostly empty) box may cause a parallel simulation to lose atoms the first time that LIGGGHTS(R)-WSU shrink-wraps the box around the atoms.

Restrictions:

An **atom_style** and **region** must have been previously defined to use this command.

Related commands:

create_atoms, region

Default: none

delete_atoms command

Syntax:

```
delete_atoms style args keyword value ...
```

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

```
  group args = group-ID
    region args = region-ID
    overlap args = cutoff group1-ID group2-ID
      cutoff = delete one atom from pairs of atoms within the cutoff (distance units)
      group1-ID = one atom in pair must be in this group
      group2-ID = other atom in pair must be in this group
    porosity args = region-ID fraction seed
      region-ID = region within which to perform deletions
      fraction = delete this fraction of atoms
      seed = random number seed (positive integer)
```

- zero or more keyword/value pairs may be appended
- keyword = *compress* or *mol*

```
  compress value = no or yes
    mol value = no or yes
```

Examples:

```
delete_atoms group edge
delete_atoms region sphere compress no
delete_atoms overlap 0.3 all all
delete_atoms overlap 0.5 solvent colloid
delete_atoms porosity cube 0.1 482793
```

Description:

Delete the specified atoms. This command can be used to carve out voids from a block of material or to delete created atoms that are too close to each other (e.g. at a grain boundary).

For style *group*, all atoms belonging to the group are deleted.

For style *region*, all atoms in the region volume are deleted. Additional atoms can be deleted if they are in a molecule for which one or more atoms were deleted within the region; see the *mol* keyword discussion below.

For style *overlap* pairs of atoms whose distance of separation is within the specified cutoff distance are searched for, and one of the 2 atoms is deleted. Only pairs where one of the two atoms is in the first group specified and the other atom is in the second group are considered. The atom that is in the first group is the one that is deleted.

Note that it is OK for the two group IDs to be the same (e.g. group *all*), or for some atoms to be members of both groups. In these cases, either atom in the pair may be deleted. Also note that if there are atoms which are members of both groups, the only guarantee is that at the end of the deletion operation, enough deletions will have occurred that no atom pairs within the cutoff will remain (subject to the group restriction). There is no guarantee that the minimum number of atoms will be deleted, or that the same atoms will be deleted when running on different numbers of processors.

For style *porosity* a specified *fraction* of atoms are deleted within the specified region. For example, if fraction is 0.1, then 10% of the atoms will be deleted. The atoms to delete are chosen randomly. There is no guarantee that the exact fraction of atoms will be deleted, or that the same atoms will be deleted when running on different numbers of processors.

If the *compress* keyword is set to *yes*, then after atoms are deleted, then atom IDs are re-assigned so that they run from 1 to the number of atoms in the system. This is not done for molecular systems (see the atom_style command), regardless of the *compress* setting, since it would foul up the bond connectivity that has already been assigned.

It the *mol* keyword is set to *yes*, then for every atom that is deleted, all other atoms in the same molecule will also be deleted. This keyword is only used by the *region* style. It is a way to insure that entire molecules are deleted instead of only a subset of atoms in a bond or angle or dihedral interaction.

Restrictions:

The *overlap* styles requires inter-processor communication to acquire ghost atoms and build a neighbor list. This means that your system must be ready to perform a simulation before using this command (force fields setup, atom masses set, etc). Since a neighbor list is used to find overlapping atom pairs, it also means that you must define a pair style with force cutoffs greater than or equal to the desired overlap cutoff between pairs of relevant atom types, even though the pair potential will not be evaluated.

If the special_bonds command is used with a setting of 0, then a pair of bonded atoms (1-2, 1-3, or 1-4) will not appear in the neighbor list, and thus will not be considered for deletion by the *overlap* styles. You probably don't want to be deleting one atom in a bonded pair anyway.

Related commands:

create_atoms

Default:

The option defaults are compress = yes and mol = no.

delete_bonds command

Syntax:

```
delete_bonds group-ID style args keyword ...
```

- group-ID = group ID
- style = *multi* or *atom* or *bond* or *stats*

```
  multi args = none
    atom args = an atom type
    bond args = a bond type
    stats args = none
```
- zero or more keywords may be appended
- keyword = *any* or *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. 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.

For all styles, by default, an interaction is only turned off (or on) if all the atoms involved are in the specified group. See the *any* keyword to change the behavior.

Style *atom* is the same as style *multi* except that in addition, one or more of the atoms involved in the bond, interaction must also be of the specified atom type.

For style *bond*, only bonds are candidates for turn-off, and the bond must also be of the specified type.

For style *bond*, you can set the type to 0 to delete bonds that have been previously broken by a bond-breaking potential (which sets the bond type to 0 when a bond is broken); e.g. see the [bond_style quartic](#) command.

For style *stats* no interactions are turned off (or on); the status of all interactions in the specified group is simply reported. This is useful for diagnostic purposes if bonds have been turned off by a bond-breaking potential during a previous run.

The default behavior of the delete_bonds command is to turn off interactions by toggling their type to a negative value, but not to permanently remove the interaction. 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 were still turned on.

Several keywords can be appended to the argument list to alter the default behaviors.

The *any* keyword changes the requirement that all atoms in the bond must be in the specified group in order to turn-off the interaction. Instead, if any of the atoms in the interaction are in the specified group, it will be turned off (or on if the *undo* keyword is used).

The *undo* keyword inverts the delete_bonds command so that the specified bonds are turned on if they are currently turned off.

The *remove* keyword is invoked at the end of the delete_bonds operation. It causes turned-off bonds to be removed from each atom's data structure and then adjusts the global bond counts accordingly. Removal is a permanent change; removed bonds cannot be turned back on via the *undo* keyword. Removal does not alter the pairwise 1-2, 1-3, 1-4 weighting list.

The *special* keyword is invoked at the end of the delete_bonds operation, after (optional) removal. It re-computes the pairwise 1-2, 1-3, 1-4 weighting list. The weighting list computation treats turned-off bonds the same as turned-on. Thus, turned-off bonds must be removed if you wish to change the weighting list.

Note that the choice of *remove* and *special* options affects how 1-2, 1-3, 1-4 pairwise interactions will be computed across bonds that have been modified by the delete_bonds command.

Restrictions:

This command requires inter-processor communication to coordinate the deleting of bonds. This means that your system must be ready to perform a simulation before using this command (force fields setup, atom masses set, etc).

Related commands:

neigh_modify exclude, special_bonds

Default: none

dielectric command

Syntax:

```
dielectric value
```

- value = dielectric constant

Examples:

```
dielectric 2.0
```

Description:

Set the dielectric constant for Coulombic interactions (pairwise and long-range) to this value. The constant is unitless, since it is used to reduce the strength of the interactions. The value is used in the denominator of the formulas for Coulombic interactions - e.g. a value of 4.0 reduces the Coulombic interactions to 25% of their default strength. See the pair_style command for more details.

Restrictions: none

Related commands:

pair_style

Default:

```
dielectric 1.0
```

dimension command

Syntax:

```
dimension N
```

- N = 2 or 3

Examples:

```
dimension 2
```

Description:

Set the dimensionality of the simulation. By default LIGGGHTS(R)-WSU 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 Section_howto for additional instructions on how to run 2d simulations.

IMPORTANT NOTE: Some models in LIGGGHTS(R)-WSU treat particles as finite-size spheres or ellipsoids, as opposed to point particles. In 2d, the particles will still be spheres or ellipsoids, not circular disks or ellipses, meaning their moment of inertia will be the same as in 3d.

Restrictions:

This command must be used before the simulation box is defined by a read_data or create_box command.

Related commands:

fix enforce2d

Default:

```
dimension 3
```

displace_atoms command

Syntax:

```
displace_atoms group-ID style args keyword value ...
```

- group-ID = ID of group of atoms to displace
- style = *move* or *ramp* or *random* or *rotate*

```
  move args = delx dely delz
      delx,dely,delz = distance to displace in each dimension (distance units)
    ramp args = ddim dlo dhi dim clo chi
      ddim = x or y or z
      dlo,dhi = displacement distance between dlo and dhi (distance units)
      dim = x or y or z
      clo,chi = lower and upper bound of domain to displace (distance units)
    random args = dx dy dz seed
      dx,dy,dz = random displacement magnitude in each dimension (distance units)
      seed = random # seed (positive integer)
    rotate args = Px Py Pz Rx Ry Rz theta
      Px,Py,Pz = origin point of axis of rotation (distance units)
      Rx,Ry,Rz = axis of rotation vector
      theta = angle of rotation (degrees)
```

- zero or more keyword/value pairs may be appended

```
      keyword = units
        value = box or lattice
```

Examples:

```
displace_atoms top move 0 -5 0 units box
displace_atoms flow ramp x 0.0 5.0 y 2.0 20.5
```

Description:

Displace a group of atoms. This can be used to move atoms a large distance before beginning a simulation or to randomize atoms initially on a lattice. For example, in a shear simulation, an initial strain can be imposed on the system. Or two groups of atoms can be brought into closer proximity.

The *move* style displaces the group of atoms by the specified 3d distance.

The *ramp* style displaces atoms a variable amount in one dimension depending on the atom's coordinate in a (possibly) different dimension. For example, the second example command displaces atoms in the x-direction an amount between 0.0 and 5.0 distance units. Each atom's displacement depends on the fractional distance its y coordinate is between 2.0 and 20.5. Atoms with y-coordinates outside those bounds will be moved the minimum (0.0) or maximum (5.0) amount.

The *random* style independently moves each atom in the group by a random displacement, uniformly sampled from a value between -dx and +dx in the x dimension, and similarly for y and z. Random numbers are used in such a way that the displacement of a particular atom is the same, regardless of how many processors are being used.

The *rotate* style rotates each atom in the group by the angle *theta* around a rotation axis R = (Rx,Ry,Rz) that goes thru a point P = (Px,Py,Pz). The direction of rotation for the atoms around the rotation axis is consistent with the right-hand rule: if your right-hand's thumb points along R, then your fingers wrap around the axis in

the direction of positive theta.

Distance units for displacements and the origin point of the *rotate* style are determined by the setting of *box* or *lattice* for the *units* keyword. *Box* means distance units as defined by the [units](#) command - e.g. Angstroms for *real* units. *Lattice* means distance units are in lattice spacings. The [lattice](#) command must have been previously used to define the lattice spacing.

IMPORTANT NOTE: Care should be taken not to move atoms on top of other atoms. After the move, atoms are remapped into the periodic simulation box if needed, and any shrink-wrap boundary conditions (see the [boundary](#) command) are enforced which may change the box size. Other than this effect, this command does not change the size or shape of the simulation box. See the [change_box](#) command if that effect is desired.

IMPORTANT NOTE: Atoms can be moved arbitrarily long distances by this command. If the simulation box is non-periodic and shrink-wrapped (see the [boundary](#) command), this can change its size or shape. This is not a problem, except that the mapping of processors to the simulation box is not changed by this command from its initial 3d configuration; see the [processors](#) command. Thus, if the box size/shape changes dramatically, the mapping of processors to the simulation box may not end up as optimal as the initial mapping attempted to be.

Restrictions:

You cannot rotate around any rotation vector except the z-axis for a 2d simulation.

Related commands:

[lattice](#), [change_box](#), [fix_move](#)

Default:

The option defaults are units = lattice.

dump custom/vtk 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 = *custom/vtk*
- N = dump every this many timesteps
- file = name of file to write dump info to
- args = list of arguments for a particular style

```
  custom/vtk args = list of atom attributes
      possible attributes = id, mol, id_multisphere, type, element, mass, density, rho, p
                            x, y, z, xs, ys, zs, xu, yu, zu,
                            xsu, ysu, zsu, ix, iy, iz,
                            vx, vy, vz, fx, fy, fz,
                            q, mux, muy, muz, mu,
                            radius, diameter, omegax, omegay, omegaz,
                            angmomx, angmomy, angmomz, tqx, tqy, tqz,
                            c_ID, c_ID[N], f_ID, f_ID[N], v_name

          id = atom ID
          mol = molecule ID
          id_multisphere = ID of multisphere body
          type = atom type
          element = name of atom element, as defined by dump_modify command
          mass = atom mass
          x,y,z = unscaled atom coordinates
          xs,ys,zs = scaled atom coordinates
          xu,yu,zu = unwrapped atom coordinates
          xsu,ysu,zsu = scaled unwrapped atom coordinates
          ix,iy,iz = box image that the atom is in
          vx,vy,vz = atom velocities
          fx,fy,fz = forces on atoms
          q = atom charge
          mux,muy,muz = orientation of dipole moment of atom
          mu = magnitude of dipole moment of atom
          radius,diameter = radius,diameter of spherical particle
          omegax,omegay,omegaz = angular velocity of spherical particle
          angmomx,angmomy,angmomz = angular momentum of aspherical particle
          tqx,tqy,tqz = torque on finite-size particles
          c_ID = per-atom vector calculated by a compute with ID
          c_ID[N] = Nth column of per-atom array calculated by a compute with ID
          f_ID = per-atom vector calculated by a fix with ID
          f_ID[N] = Nth column of per-atom array calculated by a fix with ID
          v_name = per-atom vector calculated by an atom-style variable with name
          shapex, shapey, shapez = semi-axes for superquadric particles
          roundness1, roundness2 = roundness/blockiness parameters for superquadric particles
          quat1, quat2, quat3, quat4 = quaternion components for superquadric paricles
```

Examples:

```
dump dmpvtk all custom/vtk 100 dump*.myforce.vtk id type vx fx
dump dmpvtp flow custom/vtk 100 dump*.%.displace.vtp id type c_myD[1] c_myD[2] c_myD[3] v_ke
```

Description:

Dump a snapshot of atom quantities to one or more files every N timesteps. The timesteps on which dump output is written can also be controlled by a variable; see the dump_modify every command for details.

Only information for atoms in the specified group is dumped. The dump_modify thresh and region commands can also alter what atoms are included; see details below.

As described below, special characters ("*", "%") in the filename determine the kind of output.

IMPORTANT NOTE: Because periodic boundary conditions are enforced only on timesteps when neighbor lists are rebuilt, the coordinates of an atom written to a dump file may be slightly outside the simulation box.

IMPORTANT NOTE: Unless the dump_modify sort option is invoked, the lines of atom information written to dump files will be in an indeterminate order for each snapshot. This is even true when running on a single processor, if the atom_modify sort option is on, which it is by default. In this case atoms are re-ordered periodically during a simulation, due to spatial sorting. It is also true when running in parallel, because data for a single snapshot is collected from multiple processors, each of which owns a subset of the atoms.

For the *custom/vtk* style, sorting is off by default. See the dump_modify doc page for details.

The dimensions of the simulation box are written to a separate file for each snapshot (either in legacy VTK or XML format depending on the format of the main dump file) with the suffix *_boundingBox* appended to the given dump filename.

For an orthogonal simulation box this information is saved as a rectilinear grid (legacy .vtk or .vtr XML format).

Triclinic simulation boxes (non-orthogonal) are saved as hexahedrons in either legacy .vtk or .vtu XML format.

Style *custom/vtk* allows you to specify a list of atom attributes to be written to the dump file for each atom. Possible attributes are listed above. In contrast to the *custom* style, the attributes are rearranged to ensure correct ordering of vector components (except for computes and fixes - these have to be given in the right order) and duplicate entries are removed.

You cannot specify a quantity that is not defined for a particular simulation - such as *q* for atom style *bond*, since that atom style doesn't assign charges. Dumps occur at the very end of a timestep, so atom attributes will include effects due to fixes that are applied during the timestep. An explanation of the possible dump custom/vtk attributes is given below. Since position data is required to write VTK files "x y z" do not have to be specified explicitly.

The VTK format uses a single snapshot of the system per file, thus a wildcard "*" must be included in the filename, as discussed below. Otherwise the dump files will get overwritten with the new snapshot each time.

Dumps are performed on timesteps that are a multiple of N (including timestep 0) and on the last timestep of a minimization if the minimization converges. Note that this means a dump will not be performed on the initial timestep after the dump command is invoked, if the current timestep is not a multiple of N. This behavior can be changed via the dump_modify first command, which can also be useful if the dump command is invoked after a minimization ended on an arbitrary timestep. N can be changed between runs by using the dump_modify every command. The dump_modify every command also allows a variable to be used to determine the sequence of timesteps on which dump files are written. In this mode a dump on the first timestep of a run will also not be written unless the dump_modify first command is used.

Dump filenames can contain two wildcard characters. If a "*" character appears in the filename, then one file per snapshot is written and the "*" character is replaced with the timestep value. For example, tmp.dump*.vtk

dump custom/vtk command   111

becomes tmp.dump0.vtk, tmp.dump10000.vtk, tmp.dump20000.vtk, etc. Note that the dump_modify pad command can be used to insure all timestep numbers are the same length (e.g. 00010), which can make it easier to read a series of dump files in order with some post-processing tools.

If a "%" character appears in the filename, then each of P processors writes a portion of the dump file, and the "%" character is replaced with the processor ID from 0 to P-1 preceded by an underscore character. For example, tmp.dump%.vtp becomes tmp.dump_0.vtp, tmp.dump_1.vtp, ... tmp.dump_P-1.vtp, etc. This creates smaller files and can be a fast mode of output on parallel machines that support parallel I/O for output.

By default, P = the number of processors meaning one file per processor, but P can be set to a smaller value via the *nfile* or *fileper* keywords of the dump_modify command. These options can be the most efficient way of writing out dump files when running on large numbers of processors.

For the legacy VTK format "%" is ignored and P = 1, i.e., only processor 0 does write files.

Note that using the "*" and "%" characters together can produce a large number of small dump files!

If *dump_modify binary* is used, the dump file (or files, if "*" or "%" is also used) is written in binary format. A binary dump file will be about the same size as a text version, but will typically write out much faster.

This section explains the atom attributes that can be specified as part of the *custom/vtk* style.

The *id*, *mol*, *id_multisphere*, *type*, *element*, *mass*, *vx*, *vy*, *vz*, *fx*, *fy*, *fz*, *q* attributes are self-explanatory.

id is the atom ID. *mol* is the molecule ID, included in the data file for molecular systems. *id_multisphere* is the ID of the multisphere body that the particle belongs to (if your version supports multisphere). *type* is the atom type. *element* is typically the chemical name of an element, which you must assign to each type via the dump_modify element command. More generally, it can be any string you wish to associate with an atom type. *mass* is the atom mass. *vx*, *vy*, *vz*, *fx*, *fy*, *fz*, and *q* are components of atom velocity and force and atomic charge.

There are several options for outputting atom coordinates. The *x*, *y*, *z* attributes are used to write atom coordinates "unscaled", in the appropriate distance units (Angstroms, sigma, etc). Additionally, you can use *xs*, *ys*, *zs* if you want to also save the coordinates "scaled" to the box size, so that each value is 0.0 to 1.0. If the simulation box is triclinic (tilted), then all atom coords will still be between 0.0 and 1.0. Use *xu*, *yu*, *zu* if you want the coordinates "unwrapped" by the image flags for each atom. Unwrapped means that if the atom has passed through a periodic boundary one or more times, the value is printed for what the coordinate would be if it had not been wrapped back into the periodic box. Note that using *xu*, *yu*, *zu* means that the coordinate values may be far outside the box bounds printed with the snapshot. Using *xsu*, *ysu*, *zsu* is similar to using *xu*, *yu*, *zu*, except that the unwrapped coordinates are scaled by the box size. Atoms that have passed through a periodic boundary will have the corresponding cooordinate increased or decreased by 1.0.

The image flags can be printed directly using the *ix*, *iy*, *iz* attributes. For periodic dimensions, they specify which image of the simulation box the atom is considered to be in. An image of 0 means it is inside the box as defined. A value of 2 means add 2 box lengths to get the true value. A value of -1 means subtract 1 box length to get the true value. LIGGGHTS(R)-WSU updates these flags as atoms cross periodic boundaries during the simulation.

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

The *radius* and *diameter* attributes are specific to spherical particles that have a finite size, such as those defined with an atom style of *sphere*. For *superquadric* particles these attributes give bounding sphere radius.

The *omegax*, *omegay*, and *omegaz* attributes are specific to finite-size spherical particles that have an angular velocity. Only certain atom styles, such as *sphere* define this quantity.

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

The *tqx*, *tqy*, *tqz* attributes are for finite-size particles that can sustain a rotational torque due to interactions with other particles.

The *c_ID* and *c_ID[N]* attributes allow per-atom vectors or arrays calculated by a [compute](#) to be output. The ID in the attribute should be replaced by the actual ID of the compute that has been defined previously in the input script. See the [compute](#) command for details. There are computes for calculating the per-atom energy, stress, centro-symmetry parameter, and coordination number of individual atoms.

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

If *c_ID* is used as an attribute, then the per-atom vector calculated by the compute is printed. If *c_ID[N]* is used, then N must be in the range from 1-M, which will print the Nth column of the M-length per-atom array calculated by the compute.

The *f_ID* and *f_ID[N]* attributes allow vector or array per-atom quantities calculated by a [fix](#) to be output. The ID in the attribute should be replaced by the actual ID of the fix that has been defined previously in the input script. The [fix ave/atom](#) command is one that calculates per-atom quantities. Since it can time-average per-atom quantities produced by any [compute](#), [fix](#), or atom-style [variable](#), this allows those time-averaged results to be written to a dump file.

If *f_ID* is used as a attribute, then the per-atom vector calculated by the fix is printed. If *f_ID[N]* is used, then N must be in the range from 1-M, which will print the Nth column of the M-length per-atom array calculated by the fix.

The *v_name* attribute allows per-atom vectors calculated by a [variable](#) to be output. The name in the attribute should be replaced by the actual name of the variable that has been defined previously in the input script. Only an atom-style variable can be referenced, since it is the only style that generates per-atom values. Variables of style *atom* can reference individual atom attributes, per-atom atom attributes, thermodynamic keywords, or invoke other computes, fixes, or variables when they are evaluated, so this is a very general means of creating quantities to output to a dump file.

The *shapex, shapey, shapez, roundness1, roundness2, quat1, quat2, quat3, quat4* attributes are available only for *superquadric* particles and hence require this [atom_style](#)

See [Section_modify](#) of the manual for information on how to add new compute and fix styles to LIGGGHTS(R)-WSU to calculate per-atom quantities which could then be output into dump files.

Restrictions:

The *custom/vtk* style does not support writing of gzipped dump files.

To be able to use *custom/vtk*, you have to link to the VTK library, please adapt your Makefile accordingly. You must compile LIGGGHTS(R)-WSU with the -DLAMMPS_VTK option - see the [Making LIGGGHTS(R)-WSU](#) section of the documentation.

The *custom/vtk* dump style neither supports buffering nor custom format strings.

Related commands:

dump, dump image, dump_modify, undump

Default:

By default, files are written in ASCII format. If the file extension is not one of .vtk, .vtp or .vtu, the legacy VTK file format is used.

dump command

dump image command

dump movie 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 *atom/vtk* or *xyz* or *image* or *local* or *custom* or *mesh/stl* or *mesh/vtk* or *mesh/vtk* or *decomposition/vtk* or *euler/vtk*
- 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
  atom/vtk args = none
  xyz args = none

image args = discussed on dump image doc page

mesh/stl args = 'local' or 'ghost' or 'all' or 'region' or any ID of a fix mesh/surface
      region values = ID for region threshold
  mesh/vtk args =  zero or more keyword/ value pairs and one or more dump-identifiers
      keywords = output
      output values = face or interpolate
      dump-identifier = 'stress' or 'id' or 'wear' or 'vel' or 'stresscomponents' or 'owne
  euler/vtk args = none
  decomposition/vtk args = none

local args = list of local attributes
    possible attributes = index, c_ID, c_ID[N], f_ID, f_ID[N]
      index = enumeration of local values
      c_ID = local vector calculated by a compute with ID
      c_ID[N] = Nth column of local array calculated by a compute with ID
      f_ID = local vector calculated by a fix with ID
      f_ID[N] = Nth column of local array calculated by a fix with ID

custom args = list of atom attributes
    possible attributes = id, mol, id_multisphere , type, element, mass,
                          x, y, z, xs, ys, zs, xu, yu, zu,
                          xsu, ysu, zsu, ix, iy, iz,
                          vx, vy, vz, fx, fy, fz,
                          q, mux, muy, muz, mu,
                          radius, diameter, omegax, omegay, omegaz,
                          angmomx, angmomy, angmomz, tqx, tqy, tqz,
                          c_ID, c_ID[N], f_ID, f_ID[N], v_name

        id = atom ID
        mol = molecule ID
        id_multisphere = ID of multisphere body
        type = atom type
        element = name of atom element, as defined by dump_modify command
        mass = atom mass
```

```
        x,y,z = unscaled atom coordinates
        xs,ys,zs = scaled atom coordinates
        xu,yu,zu = unwrapped atom coordinates
        xsu,ysu,zsu = scaled unwrapped atom coordinates
        ix,iy,iz = box image that the atom is in
        vx,vy,vz = atom velocities
        fx,fy,fz = forces on atoms
        q = atom charge
        mux,muy,muz = orientation of dipole moment of atom
        mu = magnitude of dipole moment of atom
        radius,diameter = radius,diameter of spherical particle
        omegax,omegay,omegaz = angular velocity of spherical particle
        angmomx,angmomy,angmomz = angular momentum of aspherical particle
        tqx,tqy,tqz = torque on finite-size particles
        c_ID = per-atom vector calculated by a compute with ID
        c_ID[N] = Nth column of per-atom array calculated by a compute with ID
        f_ID = per-atom vector calculated by a fix with ID
        f_ID[N] = Nth column of per-atom array calculated by a fix with ID
        v_name = per-atom vector calculated by an atom-style variable with name
```

Examples:

```
dump myDump all atom 100 dump.atom
dump 2 subgroup atom 50 dump.run.bin
dump 4a all custom 100 dump.myforce.* id type x y vx fx
dump 4b flow custom 100 dump.%.myforce id type c_myF[3] v_ke
```

Description:

Dump a snapshot of atom quantities to one or more files every N timesteps in one of several styles. The *image* style is the exception; it creates a JPG or PPM image file of the atom configuration every N timesteps, as discussed on the dump image doc page. The timesteps on which dump output is written can also be controlled by a variable; see the dump_modify every command for details.

Only information for atoms in the specified group is dumped. The dump_modify thresh and region commands can also alter what atoms are included. Not all styles support all these options; see details below.

As described below, the filename determines the kind of output (text or binary or gzipped, one big file or one per timestep, one big file or multiple smaller files).

IMPORTANT NOTE: Because periodic boundary conditions are enforced only on timesteps when neighbor lists are rebuilt, the coordinates of an atom written to a dump file may be slightly outside the simulation box.

IMPORTANT NOTE: Unless the dump_modify sort option is invoked, the lines of atom information written to dump files (typically one line per atom) will be in an indeterminate order for each snapshot. This is even true when running on a single processor, if the atom_modify sort option is on, which it is by default. In this case atoms are re-ordered periodically during a simulation, due to spatial sorting. It is also true when running in parallel, because data for a single snapshot is collected from multiple processors, each of which owns a subset of the atoms.

For the *atom*, *custom*, and *local* styles, sorting is off by default. For the *xyz* style, sorting by atom ID is on by default. See the dump_modify doc page for details.

The *style* keyword determines what atom quantities are written to the file and in what format. Settings made via the dump_modify command can also alter the format of individual values and the file itself.

The *atom*, *local*, and *custom* styles create files in a simple text format that is self-explanatory when viewing a dump file.

dump movie command   116

For post-processing purposes the *atom*, *local*, and *custom* text files are self-describing in the following sense.

The dimensions of the simulation box are included in each snapshot. For an orthogonal simulation box this information is is formatted as:

```
ITEM: BOX BOUNDS xx yy zz
xlo xhi
ylo yhi
zlo zhi
```

where xlo,xhi are the maximum extents of the simulation box in the x-dimension, and similarly for y and z. The "xx yy zz" represent 6 characters that encode the style of boundary for each of the 6 simulation box boundaries (xlo,xhi and ylo,yhi and zlo,zhi). Each of the 6 characters is either p = periodic, f = fixed, s = shrink wrap, or m = shrink wrapped with a minimum value. See the **boundary** command for details.

For triclinic simulation boxes (non-orthogonal), an orthogonal bounding box which encloses the triclinic simulation box is output, along with the 3 tilt factors (xy, xz, yz) of the triclinic box, formatted as follows:

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

The presence of the text "xy xz yz" in the ITEM line indicates that the 3 tilt factors will be included on each of the 3 following lines. This bounding box is convenient for many visualization programs. The meaning of the 6 character flags for "xx yy zz" is the same as above.

Note that the first two numbers on each line are now xlo_bound instead of xlo, etc, since they repesent a bounding box. See **this section** of the doc pages for a geometric description of triclinic boxes, as defined by LIGGGHTS(R)-WSU, simple formulas for how the 6 bounding box extents (xlo_bound,xhi_bound,etc) are calculated from the triclinic parameters, and how to transform those parameters to and from other commonly used triclinic representations.

The "ITEM: ATOMS" line in each snapshot lists column descriptors for the per-atom lines that follow. For example, the descriptors would be "id type xs ys zs" for the default *atom* style, and would be the atom attributes you specify in the dump command for the *custom* style.

For style *atom*, atom coordinates are written to the file, along with the atom ID and atom type. By default, atom coords are written in a scaled format (from 0 to 1). I.e. an x value of 0.25 means the atom is at a location 1/4 of the distance from xlo to xhi of the box boundaries. The format can be changed to unscaled coords via the **dump_modify** settings. Image flags can also be added for each atom via dump_modify.

For style *atom/vtk*, atom coordinates, velocity, rotational velocity, force, atom ID, atom radius and atom type are written to VTK files. Note that you have to link against VTK libraries to use this functionality.

Style *custom* allows you to specify a list of atom attributes to be written to the dump file for each atom. Possible attributes are listed above and will appear in the order specified. You cannot specify a quantity that is not defined for a particular simulation - such as *q* for atom style *bond*, since that atom style doesn't assign charges. Dumps occur at the very end of a timestep, so atom attributes will include effects due to fixes that are applied during the timestep. An explanation of the possible dump custom attributes is given below.

For style *local*, local output generated by **computes** and **fixes** is used to generate lines of output that is written to the dump file. This local data is typically calculated by each processor based on the atoms it owns, but there may be zero or more entities per atom, e.g. a list of bond distances. An explanation of the possible dump local attributes is given below. Note that by using input from the **compute property/local** command with dump local, it is possible to generate information on bonds that can be cut and pasted directly into a data file read by

dump movie command   117

the read_data command.

The *xyz* style writes XYZ files, which is a simple text-based coordinate format that many codes can read. Specifically it has a line with the number of atoms, then a comment line that is usually ignored followed by one line per atom with the atom type and the x-, y-, and z-coordinate of that atom. You can use the dump_modify element option to change the output from using the (numerical) atom type to an element name (or some other label). This will help many visualization programs to guess bonds and colors.

The *mesh/stl* style dumps active STL geometries defined via fix mesh commands into the specified file. If you do not supply the optional list of mesh IDs, all meshes are dumped, irrespective of whether they are used in a fix wall/gran command or not. By specifying a list of mesh IDs you can explicitly choose which meshes to dump. The group-ID is ignored, because the command is not applied to particles, but to mesh geometries. With keywords 'local', 'owned' or 'ghost' you can decide which parts of the parallel meshes you want to dump (default is 'local'). If the multiprocessor option is not used (no '%' in filename), data is gathered from all processors, so using the default will output the whole mesh data across all processors.

Examples:

```
dump stl1 all mesh/stl 300 post/dump*.stl
dump stl2 all mesh/stl 300 post/dump_proc%_local*.stl local
dump stl3 all mesh/stl 300 post/dump_proc%_ghost*.stl ghost
dump stl4 all mesh/stl 300 post/dump_proc_all_ghost*.stl ghost
```

The first command will write one file per time-step containing the complete mesh. The second command will output one file per time-step per processor containing the local (owned) mesh elements of each processor. The third command will output one file per time-step per processor containing the ghost (corona) mesh elements of each processor. The third command will output one file per time-step containing the ghost (corona) mesh elements of all processors.

With the *region* keyword, just those mesh element where the element center (arithmetic average of all nodes) is in the specified region, will be dumped.

This dump is especially useful if a fix move/mesh is registered. If the position of the mesh is changed over time and you want to dump one file for each dump timestep for post-processing together with the particle data, you should use a filename like 'mydumpfile*.stl'. Note: This series of files can then be post-processed together with the particle dump file converted to VTK in Paraview , www.paraview.org

By providing any ID (or a list of IDs) of fix mesh/surface commands, you can specify which meshes to dump. If no meshes are specified, all meshes used in the simulation are dumped.

The *mesh/vtk* style can be used to dump active mesh geometries defined via fix mesh commands to a series of VTK files. Different keywords can be used to dump the per-triangle stress (force magnitude / element area), id, velocity, wear, stress components (fx / element area, fy / element area, fz / element area), area (area of each element) or the process which owns the element (visulatisation of the parallel decomposition) into the specified file using a VTK file format. The list of mesh IDs is optional. As with the stl style, all active meshes are dumped if you do not supply the optional list of mesh IDs. By specifying list of mesh IDs you can explicitly choose which meshes to dump. The group-ID is ignored. Again, a series of files can be post-processed in Paraview , www.paraview.org Most keywords as used for the *mesh/vtk* style are self-explanatory. Keyword *output* controls if the data is written in a per-face manner or as interpolated values to VTK. Keywords *aedges* and *acorners* dump the number of active edges/corners per face. Keyword *nneighs* dumps the number of face neighbors LIGGGHTS(R)-WSU has recognized for each face.

By providing any ID (or a list of IDs) of fix mesh/surface commands, you can specify which meshes to dump. If no meshes are specified, all meshes used in the simulation are dumped.

dump movie command   118

The *euler/vtk* style dumps the output of a [fix ave/euler](#) command into a series of VTK files. No further args are expected.

The *decomposition/vtk* style dumps the processor grid decomposition into a series of VTK files. No further args are expected.

Dumps are performed on timesteps that are a multiple of N (including timestep 0) and on the last timestep of a minimization if the minimization converges. Note that this means a dump will not be performed on the initial timestep after the dump command is invoked, if the current timestep is not a multiple of N. This behavior can be changed via the [dump_modify first](#) command, which can also be useful if the dump command is invoked after a minimization ended on an arbitrary timestep. N can be changed between runs by using the [dump_modify every](#) command. The [dump_modify every](#) command also allows a variable to be used to determine the sequence of timesteps on which dump files are written. In this mode a dump on the first timestep of a run will also not be written unless the [dump_modify first](#) command is used.

The specified filename determines how the dump file(s) is written. The default is to write one large text file, which is opened when the dump command is invoked and closed when an [undump](#) command is used or when LIGGGHTS(R)-WSU exits.

Dump filenames can contain two wildcard characters. If a "*" character appears in the filename, then one file per snapshot is written and the "*" character is replaced with the timestep value. For example, tmp.dump.* becomes tmp.dump.0, tmp.dump.10000, tmp.dump.20000, etc. Note that the [dump_modify pad](#) command can be used to insure all timestep numbers are the same length (e.g. 00010), which can make it easier to read a series of dump files in order with some post-processing tools.

If a "%" character appears in the filename, then each of P processors writes a portion of the dump file, and the "%" character is replaced with the processor ID from 0 to P-1. For example, tmp.dump.% becomes tmp.dump.0, tmp.dump.1, ... tmp.dump.P-1, etc. This creates smaller files and can be a fast mode of output on parallel machines that support parallel I/O for output. This option is not available for the *xyz* style.

By default, P = the number of processors meaning one file per processor, but P can be set to a smaller value via the *nfile* or *fileper* keywords of the [dump_modify](#) command. These options can be the most efficient way of writing out dump files when running on large numbers of processors.

Note that using the "*" and "%" characters together can produce a large number of small dump files!

If the filename ends with ".bin", the dump file (or files, if "*" or "%" is also used) is written in binary format. A binary dump file will be about the same size as a text version, but will typically write out much faster. This option is only available for the *atom* and *custom* styles.

If the filename ends with ".gz", the dump file (or files, if "*" or "%" is also used) is written in gzipped format. A gzipped dump file will be about 3x smaller than the text version, but will also take longer to write.

This section explains the local attributes that can be specified as part of the *local* style.

The *index* attribute can be used to generate an index number from 1 to N for each line written into the dump file, where N is the total number of local datums from all processors, or lines of output that will appear in the snapshot. Note that because data from different processors depend on what atoms they currently own, and atoms migrate between processor, there is no guarantee that the same index will be used for the same info (e.g. a particular bond) in successive snapshots.

The *c_ID* and *c_ID[N]* attributes allow local vectors or arrays calculated by a [compute](#) to be output. The ID in the attribute should be replaced by the actual ID of the compute that has been defined previously in the input script. See the [compute](#) command for details. There are computes for calculating local information such as

indices, types, and energies for bonds.

Note that computes which calculate global or per-atom quantities, as opposed to local quantities, cannot be output in a dump local command. Instead, global quantities can be output by the thermo_style custom command, and per-atom quantities can be output by the dump custom command.

If *c_ID* is used as a attribute, then the local vector calculated by the compute is printed. If *c_ID[N]* is used, then N must be in the range from 1-M, which will print the Nth column of the M-length local array calculated by the compute.

The *f_ID* and *f_ID[N]* attributes allow local vectors or arrays calculated by a fix to be output. The ID in the attribute should be replaced by the actual ID of the fix that has been defined previously in the input script.

If *f_ID* is used as a attribute, then the local vector calculated by the fix is printed. If *f_ID[N]* is used, then N must be in the range from 1-M, which will print the Nth column of the M-length local array calculated by the fix.

Here is an example of how to dump bond info for a system, including the distance and energy of each bond:

```
compute 1 all property/local batom1 batom2 btype
compute 2 all bond/local dist eng
dump 1 all local 1000 tmp.dump index c_1[1] c_1[2] c_1[3] c_2[1] c_2[2]
```

This section explains the atom attributes that can be specified as part of the *custom* and style.

The *id*, *mol*, *type*, *element*, *mass*, *vx*, *vy*, *vz*, *fx*, *fy*, *fz*, *q* attributes are self-explanatory.

Id is the atom ID. *Mol* is the molecule ID, included in the data file for molecular systems. *id_multisphere* is the ID of the multisphere body that the particle belongs to (if your version supports multisphere). *Type* is the atom type. *Element* is typically the chemical name of an element, which you must assign to each type via the dump_modify element command. More generally, it can be any string you wish to associated with an atom type. *Mass* is the atom mass. *Vx*, *vy*, *vz*, *fx*, *fy*, *fz*, and *q* are components of atom velocity and force and atomic charge.

There are several options for outputting atom coordinates. The *x*, *y*, *z* attributes write atom coordinates "unscaled", in the appropriate distance units (Angstroms, sigma, etc). Use *xs*, *ys*, *zs* if you want the coordinates "scaled" to the box size, so that each value is 0.0 to 1.0. If the simulation box is triclinic (tilted), then all atom coords will still be between 0.0 and 1.0. Use *xu*, *yu*, *zu* if you want the coordinates "unwrapped" by the image flags for each atom. Unwrapped means that if the atom has passed thru a periodic boundary one or more times, the value is printed for what the coordinate would be if it had not been wrapped back into the periodic box. Note that using *xu*, *yu*, *zu* means that the coordinate values may be far outside the box bounds printed with the snapshot. Using *xsu*, *ysu*, *zsu* is similar to using *xu*, *yu*, *zu*, except that the unwrapped coordinates are scaled by the box size. Atoms that have passed through a periodic boundary will have the corresponding cooordinate increased or decreased by 1.0.

The image flags can be printed directly using the *ix*, *iy*, *iz* attributes. For periodic dimensions, they specify which image of the simulation box the atom is considered to be in. An image of 0 means it is inside the box as defined. A value of 2 means add 2 box lengths to get the true value. A value of -1 means subtract 1 box length to get the true value. LIGGGHTS(R)-WSU updates these flags as atoms cross periodic boundaries during the simulation.

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

The *radius* and *diameter* attributes are specific to spherical particles that have a finite size, such as those defined with an atom style of *sphere*.

The *omegax*, *omegay*, and *omegaz* attributes are specific to finite-size spherical particles that have an angular velocity. Only certain atom styles, such as *sphere* define this quantity.

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

The *tqx*, *tqy*, *tqz* attributes are for finite-size particles that can sustain a rotational torque due to interactions with other particles.

The *c_ID* and *c_ID[N]* attributes allow per-atom vectors or arrays calculated by a [compute](#) to be output. The ID in the attribute should be replaced by the actual ID of the compute that has been defined previously in the input script. See the [compute](#) command for details. There are computes for calculating the per-atom energy, stress, centro-symmetry parameter, and coordination number of individual atoms.

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

If *c_ID* is used as a attribute, then the per-atom vector calculated by the compute is printed. If *c_ID[N]* is used, then N must be in the range from 1-M, which will print the Nth column of the M-length per-atom array calculated by the compute.

The *f_ID* and *f_ID[N]* attributes allow vector or array per-atom quantities calculated by a [fix](#) to be output. The ID in the attribute should be replaced by the actual ID of the fix that has been defined previously in the input script. The [fix ave/atom](#) command is one that calculates per-atom quantities. Since it can time-average per-atom quantities produced by any [compute](#), [fix](#), or atom-style [variable](#), this allows those time-averaged results to be written to a dump file.

If *f_ID* is used as a attribute, then the per-atom vector calculated by the fix is printed. If *f_ID[N]* is used, then N must be in the range from 1-M, which will print the Nth column of the M-length per-atom array calculated by the fix.

The *v_name* attribute allows per-atom vectors calculated by a [variable](#) to be output. The name in the attribute should be replaced by the actual name of the variable that has been defined previously in the input script. Only an atom-style variable can be referenced, since it is the only style that generates per-atom values. Variables of style *atom* can reference individual atom attributes, per-atom atom attributes, thermodynamic keywords, or invoke other computes, fixes, or variables when they are evaluated, so this is a very general means of creating quantities to output to a dump file.

See [Section_modify](#) of the manual for information on how to add new compute and fix styles to LIGGGHTS(R)-WSU to calculate per-atom quantities which could then be output into dump files.

Restrictions:

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

To be able to use *atom/vtk*, you have to link to VTK libraries, please adapt your Makefile accordingly.

Related commands:

dump image, dump_modify, undump

Default:

The defaults for the image style are listed on the dump image doc page.

dump image, dump_modify, undump

dump image command

dump movie command

Syntax:

```
dump ID group-ID style N file color diameter keyword value ...
```

- ID = user-assigned name for the dump
- group-ID = ID of the group of atoms to be imaged
- style = *image* or *movie* = style of dump command (other styles *atom* or *cfg* or *dcd* or *xtc* or *xyz* or *local* or *custom* are discussed on the dump doc page)
- N = dump every this many timesteps
- file = name of file to write image to
- color = atom attribute that determines color of each atom
- diameter = atom attribute that determines size of each atom
- zero or more keyword/value pairs may be appended
- keyword = *adiam* or *atom* or *bond* or *size* or *view* or *center* or *up* or *zoom* or *persp* or *box* or *axes* or *shiny* or *ssao*

```
adiam value = number = numeric value for atom diameter (distance units)
  atom = yes/no = do or do not draw atoms
  bond values = color width = color and width of bonds
    color = atom or type or none
    width = number or atom or type or none
      number = numeric value for bond width (distance units)
  size values = width height = size of images
    width = width of image in # of pixels
    height = height of image in # of pixels
  view values = theta phi = view of simulation box
    theta = view angle from +z axis (degrees)
    phi = azimuthal view angle (degrees)
    theta or phi can be a variable (see below)
  center values = flag Cx Cy Cz = center point of image
    flag = "s" for static, "d" for dynamic
    Cx,Cy,Cz = center point of image as fraction of box dimension (0.5 = center of box)
    Cx,Cy,Cz can be variables (see below)
  up values = Ux Uy Uz = direction that is "up" in image
    Ux,Uy,Uz = components of up vector
    Ux,Uy,Uz can be variables (see below)
  zoom value = zfactor = size that simulation box appears in image
    zfactor = scale image size by factor > 1 to enlarge, factor <1 to shrink
    zfactor can be a variable (see below)
  persp value = pfactor = amount of "perspective" in image
    pfactor = amount of perspective (0 = none,  <1 = some, > 1 = highly skewed)
    pfactor can be a variable (see below)
  box values = yes/no diam = draw outline of simulation box
    yes/no = do or do not draw simulation box lines
    diam = diameter of box lines as fraction of shortest box length
  axes values = yes/no length diam = draw xyz axes
    yes/no = do or do not draw xyz axes lines next to simulation box
    length = length of axes lines as fraction of respective box lengths
    diam = diameter of axes lines as fraction of shortest box length
  shiny value = sfactor = shinyness of spheres and cylinders
    sfactor = shinyness of spheres and cylinders from 0.0 to 1.0
  ssao value = yes/no seed dfactor = SSAO depth shading
    yes/no = turn depth shading on/off
    seed = random # seed (positive integer)
```

```
          dfactor = strength of shading from 0.0 to 1.0
```

Examples:

```
dump d0 all image 100 dump.*.jpg type type
dump d1 mobile image 500 snap.*.png element element ssao yes 4539 0.6
dump d2 all image 200 img-*.ppm type type zoom 2.5 adiam 1.5 size 1280 720
dump m0 all movie 1000 movie.mpg type type size 640 480
dump m1 all movie 1000 movie.avi type type size 640 480
dump m2 all movie 100 movie.m4v type type zoom 1.8 adiam v_value size 1280 720
```

Description:

Dump a high-quality rendered image of the atom configuration every N timesteps and save the images either as a sequence of JPG or PNG, or PPM files, or as a single movie file. The options for this command as well as the dump_modify command control what is included in the image or movie and how it appears. A series of such images can easily be manually converted into an animated movie of your simulation or the process can be automated without writing the intermediate files using the dump movie style; see further details below. Other dump styles store snapshots of numerical data asociated with atoms in various formats, as discussed on the dump doc page.

Note that a set of images or a movie can be made after a simulation has been run, using the rerun command to read snapshots from an existing dump file, and using these dump commands in the rerun script to generate the images/movie.

Here are two sample images, rendered as 1024x1024 JPG files. Click to see the full-size images:
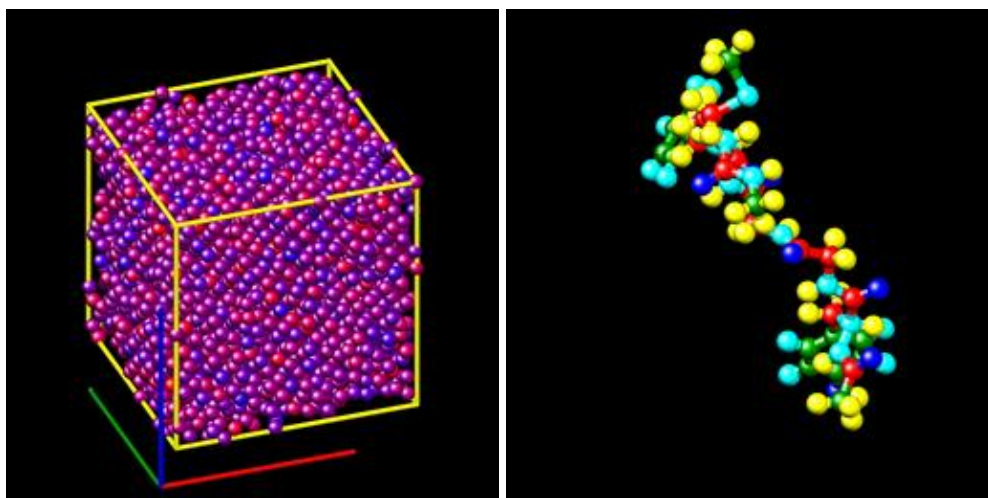
Only atoms in the specified group are rendered in the image. The dump_modify region and thresh commands can also alter what atoms are included in the image.

The filename suffix determines whether a JPEG, PNG, or PPM file is created with the *image* dump style. If the suffix is ".jpg" or ".jpeg", then a JPEG format file is created, if the suffix is ".png", then a PNG format is created, else a PPM (aka NETPBM) format file is created. The JPG and PNG files are binary; PPM has a text mode header followed by binary data. JPG images have lossy compression; PNG has lossless compression; and PPM files are uncompressed but can be compressed with gzip, if LIGGGHTS(R)-WSU has been compiled with -DLAMMPS_GZIP and a ".gz" suffix is used.

Similarly, the format of the resulting movie is chosen with the *movie* dump style. This is handled by the underlying FFmpeg converter and thus details have to be looked up in the FFmpeg documentation. Typical examples are: .avi, .mpg, .m4v, .mp4, .mkv, .flv, .mov, .gif Additional settings of the movie compression like bitrate and framerate can be set using the dump_modify command.

dump movie command   **124**

To write out JPEG and PNG format files, you must build LIGGGHTS(R)-WSU with support for the corresponding JPEG or PNG library. To convert images into movies, LIGGGHTS(R)-WSU has to be compiled with the -DLAMMPS_FFMPEG flag. See this section of the manual for instructions on how to do this.

IMPORTANT NOTE: Because periodic boundary conditions are enforced only on timesteps when neighbor lists are rebuilt, the coordinates of an atom in the image may be slightly outside the simulation box.

Dumps are performed on timesteps that are a multiple of N (including timestep 0) and on the last timestep of a minimization if the minimization converges. Note that this means a dump will not be performed on the initial timestep after the dump command is invoked, if the current timestep is not a multiple of N. This behavior can be changed via the dump_modify first command, which can be useful if the dump command is invoked after a minimization ended on an arbitrary timestep. N can be changed between runs by using the dump_modify every command.

Dump *image* filenames must contain a wildcard character "*", so that one image file per snapshot is written. The "*" character is replaced with the timestep value. For example, tmp.dump.*.jpg becomes tmp.dump.0.jpg, tmp.dump.10000.jpg, tmp.dump.20000.jpg, etc. Note that the dump_modify pad command can be used to insure all timestep numbers are the same length (e.g. 00010), which can make it easier to convert a series of images into a movie in the correct ordering.

Dump *movie* filenames on the other hand, must not have any wildcard character since only one file combining all images into a single movie will be written by the movie encoder.

The *color* and *diameter* settings determine the color and size of atoms rendered in the image. They can be any atom attribute defined for the dump custom command, including *type* and *element*. This includes per-atom quantities calculated by a compute, fix, or variable, which are prefixed by "c_", "f_", or "v_" respectively. Note that the *diameter* setting can be overridden with a numeric value by the optional *adiam* keyword, in which case you can specify the *diameter* setting with any valid atom attribute.

If *type* is specified for the *color* setting, then the color of each atom is determined by its atom type. By default the mapping of types to colors is as follows:

- type 1 = red
- type 2 = green
- type 3 = blue
- type 4 = yellow
- type 5 = aqua
- type 6 = cyan

and repeats itself for types > 6. This mapping can be changed by the dump_modify acolor command.

If *type* is specified for the *diameter* setting then the diameter of each atom is determined by its atom type. By default all types have diameter 1.0. This mapping can be changed by the dump_modify adiam command.

If *element* is specified for the *color* and/or *diameter* setting, then the color and/or diameter of each atom is determined by which element it is, which in turn is specified by the element-to-type mapping specified by the "dump_modify element" command. By default every atom type is C (carbon). Every element has a color and diameter associated with it, which is the same as the colors and sizes used by the AtomEye visualization package.

If other atom attributes are used for the *color* or *diameter* settings, they are interpreted in the following way.

If "vx", for example, is used as the *color* setting, then the color of the atom will depend on the x-component of its velocity. The association of a per-atom value with a specific color is determined by a "color map", which can be specified via the dump_modify command. The basic idea is that the atom-attribute will be within a range of values, and every value within the range is mapped to a specific color. Depending on how the color map is defined, that mapping can take place via interpolation so that a value of -3.2 is halfway between "red" and "blue", or discretely so that the value of -3.2 is "orange".

If "vx", for example, is used as the *diameter* setting, then the atom will be rendered using the x-component of its velocity as the diameter. If the per-atom value <= 0.0, them the atom will not be drawn. Note that finite-size spherical particles, as defined by atom_style sphere define a per-particle radius or diameter, which can be used as the *diameter* setting.

The various kewords listed above control how the image is rendered. As listed below, all of the keywords have defaults, most of which you will likely not need to change. The dump modify also has options specific to the dump image style, particularly for assigning colors to atoms, bonds, and other image features.

The *adiam* keyword allows you to override the *diameter* setting to a per-atom attribute with a specified numeric value. All atoms will be drawn with that diameter, e.g. 1.5, which is in whatever distance units the input script defines, e.g. Angstroms.

The *atom* keyword allow you to turn off the drawing of all atoms, if the specified value is *no*.

The *bond* keyword allows to you to alter how bonds are drawn. A bond is only drawn if both atoms in the bond are being drawn due to being in the specified group and due to other selection criteria (e.g. region, threshhold settings of the dump_modify command). By default, bonds are drawn if they are defined in the input data file as read by the read_data command. Using *none* for both the bond *color* and *width* value will turn off the drawing of all bonds.

If *atom* is specified for the bond *color* value, then each bond is drawn in 2 halves, with the color of each half being the color of the atom at that end of the bond.

If *type* is specified for the *color* value, then the color of each bond is determined by its bond type. By default the mapping of bond types to colors is as follows:

- type 1 = red
- type 2 = green
- type 3 = blue
- type 4 = yellow
- type 5 = aqua
- type 6 = cyan

and repeats itself for bond types > 6. This mapping can be changed by the dump_modify bcolor command.

The bond *width* value can be a numeric value or *atom* or *type* (or *none* as indicated above).

If a numeric value is specified, then all bonds will be drawn as cylinders with that diameter, e.g. 1.0, which is in whatever distance units the input script defines, e.g. Angstroms.

If *atom* is specified for the *width* value, then each bond will be drawn with a width corresponding to the minimum diameter of the 2 atoms in the bond.

If *type* is specified for the *width* value then the diameter of each bond is determined by its bond type. By default all types have diameter 0.5. This mapping can be changed by the dump_modify bdiam command.

---

dump movie command
126

The *size* keyword sets the width and height of the created images, i.e. the number of pixels in each direction.

The *view*, *center*, *up*, *zoom*, and *persp* values determine how 3d simulation space is mapped to the 2d plane of the image. Basically they control how the simulation box appears in the image.

All of the *view*, *center*, *up*, *zoom*, and *persp* values can be specified as numeric quantities, whose meaning is explained below. Any of them can also be specified as an <u>equal-style variable</u>, by using v_name as the value, where "name" is the variable name. In this case the variable will be evaluated on the timestep each image is created to create a new value. If the equal-style variable is time-dependent, this is a means of changing the way the simulation box appears from image to image, effectively doing a pan or fly-by view of your simulation.

The *view* keyword determines the viewpoint from which the simulation box is viewed, looking towards the *center* point. The *theta* value is the vertical angle from the +z axis, and must be an angle from 0 to 180 degrees. The *phi* value is an azimuthal angle around the z axis and can be positive or negative. A value of 0.0 is a view along the +x axis, towards the *center* point. If *theta* or *phi* are specified via variables, then the variable values should be in degrees.

The *center* keyword determines the point in simulation space that will be at the center of the image. *Cx*, *Cy*, and *Cz* are spefa as fractions of the box dimensions, so that (0.5,0.5,0.5) is the center of the simulation box. These values do not have to be between 0.0 and 1.0, if you want the simulation box to be offset from the center of the image. Note, however, that if you choose strange values for *Cx*, *Cy*, or *Cz* you may get a blank image. Internally, *Cx*, *Cy*, and *Cz* are converted into a point in simulation space. If *flag* is set to "s" for static, then this conversion is done once, at the time the dump command is issued. If *flag* is set to "d" for dynamic then the conversion is performed every time a new image is created. If the box size or shape is changing, this will adjust the center point in simulation space.

The *up* keyword determines what direction in simulation space will be "up" in the image. Internally it is stored as a vector that is in the plane perpendicular to the view vector implied by the *theta* and *pni* values, and which is also in the plane defined by the view vector and user-specified up vector. Thus this internal vector is computed from the user-specified *up* vector as

```
up_internal = view cross (up cross view)
```

This means the only restriction on the specified *up* vector is that it cannot be parallel to the *view* vector, implied by the *theta* and *phi* values.

The *zoom* keyword scales the size of the simulation box as it appears in the image. The default *zfactor* value of 1 should display an image mostly filled by the atoms in the simulation box. A *zfactor* > 1 will make the simulation box larger; a *zfactor* < 1 will make it smaller. *Zfactor* must be a value > 0.0.

The *persp* keyword determines how much depth perspective is present in the image. Depth perspective makes lines that are parallel in simulation space appear non-parallel in the image. A *pfactor* value of 0.0 means that parallel lines will meet at infininty (1.0/pfactor), which is an orthographic rendering with no persepctive. A *pfactor* value between 0.0 and 1.0 will introduce more perspective. A *pfactor* value > 1 will create a highly skewed image with a large amount of perspective.

IMPORTANT NOTE: The *persp* keyword is not yet supported as an option.

The *box* keyword determines how the simulation box boundaries are rendered as thin cylinders in the image. If *no* is set, then the box boundaries are not drawn and the *diam* setting is ignored. If *yes* is set, the 12 edges of the box are drawn, with a diameter that is a fraction of the shortest box length in x,y,z (for 3d) or x,y (for 2d). The color of the box boundaries can be set with the <u>dump_modify boxcolor</u> command.

The *axes* keyword determines how the coordinate axes are rendered as thin cylinders in the image. If *no* is set, then the axes are not drawn and the *length* and *diam* settings are ignored. If *yes* is set, 3 thin cylinders are drawn to represent the x,y,z axes in colors red,green,blue. The origin of these cylinders will be offset from the lower left corner of the box by 10%. The *length* setting determines how long the cylinders will be as a fraction of the respective box lengths. The *diam* setting determines their thickness as a fraction of the shortest box length in x,y,z (for 3d) or x,y (for 2d).

The *shiny* keyword determines how shiny the objects rendered in the image will appear. The *sfactor* value must be a value $0.0 <= sfactor <= 1.0$, where *sfactor* = 1 is a highly reflective surface and *sfactor* = 0 is a rough non-shiny surface.

The *ssao* keyword turns on/off a screen space ambient occlusion (SSAO) model for depth shading. If *yes* is set, then atoms further away from the viewer are darkened via a randomized process, which is perceived as depth. The calculation of this effect can increase the cost of computing the image by roughly 2x. The strength of the effect can be scaled by the *dfactor* parameter. If *no* is set, no depth shading is performed.

A series of JPG, PNG, or PPM images can be converted into a movie file and then played as a movie using commonly available tools. Using dump style *movie* automates this step and avoids the intermediate step of writing (many) image snapshot file. But LIGGGHTS(R)-WSU has to be compiled with -DLAMMPS_FFMPEG and an FFmpeg executable have to be installed.

To manually convert JPG, PNG or PPM files into an animated GIF or MPEG or other movie file you can use:

- a) Use the ImageMagick convert program.

    ```
    % convert *.jpg foo.gif
    % convert -loop 1 *.ppm foo.mpg
    ```

    Animated GIF files from ImageMagick are unoptimized. You can use a program like gifsicle to optimize and massively shrink them. MPEG files created by ImageMagick are in MPEG-1 format with rather inefficient compression and low quality.
- b) Use QuickTime.

    Select "Open Image Sequence" under the File menu Load the images into QuickTime to animate them Select "Export" under the File menu Save the movie as a QuickTime movie (*.mov) or in another format. QuickTime can generate very high quality and efficiently compressed movie files. Some of the supported formats require to buy a license and some are not readable on all platforms until specific runtime libraries are installed.
- c) Use FFmpeg

FFmpeg is a command line tool that is available on many platforms and allows extremely flexible encoding and decoding of movies.

```
cat snap.*.jpg | ffmpeg -y -f image2pipe -c:v mjpeg -i - -b:v 2000k movie.m4v
cat snap.*.ppm | ffmpeg -y -f image2pipe -c:v ppm -i - -b:v 2400k movie.avi
```

Frontends for FFmpeg exist for multiple platforms. For more information see the FFmpeg homepage

---

Play the movie:

- a) Use your browser to view an animated GIF movie.

    Select "Open File" under the File menu Load the animated GIF file
- b) Use the freely available mplayer or ffplay tool to view a movie. Both are available for multiple OSes and support a large variety of file formats and decoders.

```
% mplayer foo.mpg
% ffplay bar.avi
```
- c) Use the [Pizza.py](#) [animate tool](#), which works directly on a series of image files.

```
a = animate("foo*.jpg")
```
- d) QuickTime and other Windows- or MacOS-based media players can obviously play movie files directly. Similarly the corresponding tools bundled with Linux desktop environments, however, due to licensing issues of some of the file formats, some formats may require installing additional libraries, purchasing a license, or are not supported.

See [Section_modify](#) of the manual for information on how to add new compute and fix styles to LIGGGHTS(R)-WSU to calculate per-atom quantities which could then be output into dump files.

Restrictions:

To write JPG images, you must use the -DLAMMPS_JPEG switch when building LIGGGHTS(R)-WSU and link with a JPEG library. To write PNG images, you must use the -DLAMMPS_PNG switch when building LIGGGHTS(R)-WSU and link with a PNG library.

To write *movie* dumps, you must use the -DLAMMPS_FFMPEG switch when building LIGGGHTS(R)-WSU and have the FFmpeg executable available on the machine where LIGGGHTS(R)-WSU is being run.

See the [Making LIGGGHTS(R)-WSU](#) section of the documentation for details on how to configure and compile optional in LIGGGHTS(R)-WSU.

Related commands:

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

Default:

The defaults for the keywords are as follows:

- adiam = not specified (use diameter setting)
- atom = yes
- bond = none none (if no bonds in system)
- bond = atom 0.5 (if bonds in system)
- size = 512 512
- view = 60 30 (for 3d)
- view = 0 0 (for 2d)
- center = s 0.5 0.5 0.5
- up = 0 0 1 (for 3d)
- up = 0 1 0 (for 2d)
- zoom = 1.0
- persp = 0.0
- box = yes 0.02
- axes = no 0.0 0.0
- shiny = 1.0
- ssao = no

dump_modify command

Syntax:

```
dump_modify dump-ID keyword values ...
```

- dump-ID = ID of dump to modify
- one or more keyword/value pairs may be appended
- these keywords apply to various dump styles
- keyword = *append* or *buffer* or *element* or *every* or *fileper* or *first* or *flush* or *format* or *image* or *label* or *nfile* or *pad* or *precision* or *region* or *scale* or *sort* or *thresh* or *unwrap*

```
append arg = yes or no
  buffer arg = yes or no
  element args = E1 E2 ... EN, where N = # of atom types
    E1,...,EN = element name, e.g. C or Fe or Ga
  every arg = N
    N = dump every this many timesteps
    N can be a variable (see below)
  fileper arg = Np
    Np = write one file for every this many processors
  first arg = yes or no
  format arg = C-style format string for one line of output
  flush arg = yes or no
  image arg = yes or no
  label arg = string
    string = character string (e.g. BONDS) to use in header of dump local file
  nfile arg = Nf
    Nf = write this many files, one from each of Nf processors
  pad arg = Nchar = # of characters to convert timestep to
  precision arg = power-of-10 value from 10 to 1000000
  region arg = region-ID or "none"
  scale arg = yes or no
  sort arg = off or id or N or -N
     off = no sorting of per-atom lines within a snapshot
     id = sort per-atom lines by atom ID
     N = sort per-atom lines in ascending order by the Nth column
     -N = sort per-atom lines in descending order by the Nth column
  thresh args = attribute operation value
    attribute = same attributes (x,fy,etotal,sxx,etc) used by dump custom style
    operation = "" or ">=" or "==" or "!="
    value = numeric value to compare to
    these 3 args can be replaced by the word "none" to turn off thresholding
  unwrap arg = yes or no
```

- these keywords apply only to the *image* and *movie* styles
- keyword = *acolor* or *adiam* or *amap* or *bcolor* or *bdiam* or *backcolor* or *boxcolor* or *color* or *bitrate* or *framerate*

```
acolor args = type color
    type = atom type or range of types (see below)
    color = name of color or color1/color2/...
  adiam args = type diam
    type = atom type or range of types (see below)
    diam = diameter of atoms of that type (distance units)
  amap args = lo hi style delta N entry1 entry2 ... entryN
    lo = number or min = lower bound of range of color map
    hi = number or max = upper bound of range of color map
    style = 2 letters = "c" or "d" or "s" plus "a" or "f"
      "c" for continuous
```

```
      "d" for discrete
      "s" for sequential
      "a" for absolute
      "f" for fractional
    delta = binsize (only used for style "s", otherwise ignored)
      binsize = range is divided into bins of this width
    N = # of subsequent entries
    entry = value color (for continuous style)
      value = number or min or max = single value within range
      color = name of color used for that value
    entry = lo hi color (for discrete style)
      lo/hi = number or min or max = lower/upper bound of subset of range
      color = name of color used for that subset of values
    entry = color (for sequential style)
      color = name of color used for a bin of values
  backcolor arg = color
    color = name of color for background
  bcolor args = type color
    type = bond type or range of types (see below)
    color = name of color or color1/color2/...
  bdiam args = type diam
    type = bond type or range of types (see below)
    diam = diameter of bonds of that type (distance units)
  bitrate arg = rate
    rate = target bitrate for movie in kbps
  boxcolor arg = color
    color = name of color for box lines
  color args = name R G B
    name = name of color
    R,G,B = red/green/blue numeric values from 0.0 to 1.0
  framerate arg = fps
    fps = frames per second for movie
```

Examples:

```
dump_modify 1 format "%d %d %20.15g %g %g" scale yes
dump_modify myDump image yes scale no flush yes
dump_modify 1 region mySphere thresh x <0.0 thresh epair >= 3.2
dump_modify xtcdump precision 10000
dump_modify 1 every 1000 nfile 20
dump_modify 1 every v_myVar
dump_modify 1 amap min max cf 0.0 3 min green 0.5 yellow max blue boxcolor red
```

Description:

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

These keywords apply to various dump styles, including the [dump image](#) and [dump movie](#) styles. The description gives details.

The *append* keyword applies to all dump styles except *cfg* and *xtc* and *dcd*. It also applies only to text output files, not to binary or gzipped or image/movie files. If specified as *yes*, then dump snapshots are appended to the end of an existing dump file. If specified as *no*, then a new dump file will be created which will overwrite an existing file with the same name. This keyword can only take effect if the dump_modify command is used after the [dump](#) command, but before the first command that causes dump snapshots to be output, e.g. a [run](#) or [minimize](#) command. Once the dump file has been opened, this keyword has no further effect.

The *buffer* keyword applies only to dump styles *atom*, *custom*, *local*, and *xyz*. It also applies only to text output files, not to binary or gzipped files. If specified as *yes*, which is the default, then each processor writes

its output into an internal text buffer, which is then sent to the processor(s) which perform file writes, and written by those processors(s) as one large chunk of text. If specified as *no*, each processor sends its per-atom data in binary format to the processor(s) which perform file wirtes, and those processor(s) format and write it line by line into the output file.

The buffering mode is typically faster since each processor does the relatively expensive task of formatting the output for its own atoms. However it requires about twice the memory (per processor) for the extra buffering.

The *element* keyword applies only to the the dump *cfg*, *xyz*, and *image* styles. It associates element names (e.g. H, C, Fe) with LIGGGHTS(R)-WSU atom types. See the list of element names at the bottom of this page.

In the case of dump *cfg*, this allows the [AtomEye](#) visualization package to read the dump file and render atoms with the appropriate size and color.

In the case of dump *image*, the output images will follow the same [AtomEye](#) convention. An element name is specified for each atom type (1 to Ntype) in the simulation. The same element name can be given to multiple atom types.

In the case of *xyz* format dumps, there are no restrictions to what label can be used as an element name. Any whitespace separated text will be accepted.

The *every* keyword changes the dump frequency originally specified by the [dump](#) command to a new value. The every keyword can be specified in one of two ways. It can be a numeric value in which case it must be > 0. Or it can be an [equal-style variable](#), which should be specified as v_name, where name is the variable name.

In this case, the variable is evaluated at the beginning of a run to determine the next timestep at which a dump snapshot will be written out. On that timestep the variable will be evaluated again to determine the next timestep, etc. Thus the variable should return timestep values. See the stagger() and logfreq() and stride() math functions for [equal-style variables](#), as examples of useful functions to use in this context. Other similar math functions could easily be added as options for [equal-style variables](#). Also see the next() function, which allows use of a file-style variable which reads successive values from a file, each time the variable is evaluated. Used with the *every* keyword, if the file contains a list of ascending timesteps, you can output snapshots whenever you wish.

Note that when using the variable option with the *every* keyword, you need to use the *first* option if you want an initial snapshot written to the dump file. The *every* keyword cannot be used with the dump *dcd* style.

For example, the following commands will write snapshots at timesteps 0,10,20,30,100,200,300,1000,2000,etc:

```
variable        s equal logfreq(10,3,10)
dump            1 all atom 100 tmp.dump
dump_modify     1 every v_s first yes
```

The following commands would write snapshots at the timesteps listed in file tmp.times:

```
variable        f file tmp.times
variable        s equal next(f)
dump            1 all atom 100 tmp.dump
dump_modify     1 every v_s
```

IMPORTANT NOTE: When using a file-style variable with the *every* keyword, the file of timesteps must list

dump_modify command   132

a first timestep that is beyond the current timestep (e.g. it cannot be 0). And it must list one or more timesteps beyond the length of the run you perform. This is because the dump command will generate an error if the next timestep it reads from the file is not a value greater than the current timestep. Thus if you wanted output on steps 0,15,100 of a 100-timestep run, the file should contain the values 15,100,101 and you should also use the dump_modify first command. Any final value > 100 could be used in place of 101.

The *first* keyword determines whether a dump snapshot is written on the very first timestep after the dump command is invoked. This will always occur if the current timestep is a multiple of N, the frequency specified in the dump command, including timestep 0. But if this is not the case, a dump snapshot will only be written if the setting of this keyword is *yes*. If it is *no*, which is the default, then it will not be written.

The *flush* keyword determines whether a flush operation is invoked after a dump snapshot is written to the dump file. A flush insures the output in that file is current (no buffering by the OS), even if LIGGGHTS(R)-WSU halts before the simulation completes. Flushes cannot be performed with dump style *xtc*.

The text-based dump styles have a default C-style format string which simply specifies %d for integers and %g for real values. The *format* keyword can be used to override the default with a new C-style format string. Do not include a trailing "\n" newline character in the format string. This option has no effect on the *dcd* and *xtc* dump styles since they write binary files. Note that for the *cfg* style, the first two fields (atom id and type) are not actually written into the CFG file, though you must include formats for them in the format string.

The *fileper* keyword is documented below with the *nfile* keyword.

The *image* keyword applies only to the dump *atom* style. If the image value is *yes*, 3 flags are appended to each atom's coords which are the absolute box image of the atom in each dimension. For example, an x image flag of -2 with a normalized coord of 0.5 means the atom is in the center of the box, but has passed thru the box boundary 2 times and is really 2 box lengths to the left of its current coordinate. Note that for dump style *custom* these various values can be printed in the dump file by using the appropriate atom attributes in the dump command itself.

The *label* keyword applies only to the dump *local* style. When it writes local information, such as bond or angle topology to a dump file, it will use the specified *label* to format the header. By default this includes 2 lines:

```
ITEM: NUMBER OF ENTRIES
ITEM: ENTRIES ...
```

The word "ENTRIES" will be replaced with the string specified, e.g. BONDS or ANGLES.

The *nfile* or *fileper* keywords can be used in conjunction with the "%" wildcard character in the specified dump file name, for all dump styles except the *dcd*, *image*, *movie*, *xtc*, and *xyz* styles (for which "%" is not allowed). As explained on the dump command doc page, the "%" character causes the dump file to be written in pieces, one piece for each of P processors. By default P = the number of processors the simulation is running on. The *nfile* or *fileper* keyword can be used to set P to a smaller value, which can be more efficient when running on a large number of processors.

The *nfile* keyword sets P to the specified Nf value. For example, if Nf = 4, and the simulation is running on 100 processors, 4 files will be written, by processors 0,25,50,75. Each will collect information from itself and the next 24 processors and write it to a dump file.

For the *fileper* keyword, the specified value of Np means write one file for every Np processors. For example, if Np = 4, every 4th processor (0,4,8,12,etc) will collect information from itself and the next 3 processors and write it to a dump file.

The *pad* keyword only applies when the dump filename is specified with a wildcard "*" character which becomes the timestep. If *pad* is 0, which is the default, the timestep is converted into a string of unpadded length, e.g. 100 or 12000 or 2000000. When *pad* is specified with *Nchar* > 0, the string is padded with leading zeroes so they are all the same length = *Nchar*. For example, pad 7 would yield 0000100, 0012000, 2000000. This can be useful so that post-processing programs can easily read the files in ascending timestep order.

The *precision* keyword only applies to the dump *xtc* style. A specified value of N means that coordinates are stored to 1/N nanometer accuracy, e.g. for N = 1000, the coordinates are written to 1/1000 nanometer accuracy.

The *region* keyword only applies to the dump *custom*, *cfg*, *image*, and *movie* styles. If specified, only atoms in the region will be written to the dump file or included in the image/movie. Only one region can be applied as a filter (the last one specified). See the region command for more details. Note that a region can be defined as the "inside" or "outside" of a geometric shape, and it can be the "union" or "intersection" of a series of simpler regions.

The *scale* keyword applies only to the dump *atom* style. A scale value of *yes* means atom coords are written in normalized units from 0.0 to 1.0 in each box dimension. If the simluation box is triclinic (tilted), then all atom coords will still be between 0.0 and 1.0. A value of *no* means they are written in absolute distance units (e.g. Angstroms or sigma).

The *sort* keyword determines whether lines of per-atom output in a snapshot are sorted or not. A sort value of *off* means they will typically be written in indeterminate order, either in serial or parallel. This is the case even in serial if the atom_modify sort option is turned on, which it is by default, to improve performance. A sort value of *id* means sort the output by atom ID. A sort value of N or -N means sort the output by the value in the Nth column of per-atom info in either ascending or descending order.

The dump *local* style cannot be sorted by atom ID, since there are typically multiple lines of output per atom. Some dump styles, such as *dcd* and *xtc*, require sorting by atom ID to format the output file correctly. If multiple processors are writing the dump file, via the "%" wildcard in the dump filename, then sorting cannot be performed.

IMPORTANT NOTE: Unless it is required by the dump style, sorting dump file output requires extra overhead in terms of CPU and communication cost, as well as memory, versus unsorted output.

The *thresh* keyword only applies to the dump *custom*, *cfg*, *image*, and *movie* styles. Multiple thresholds can be specified. Specifying "none" turns off all threshold criteria. If thresholds are specified, only atoms whose attributes meet all the threshold criteria are written to the dump file or included in the image. The possible attributes that can be tested for are the same as those that can be specified in the dump custom command, with the exception of the *element* attribute, since it is not a numeric value. Note that different attributes can be output by the dump custom command than are used as threshold criteria by the dump_modify command. E.g. you can output the coordinates and stress of atoms whose energy is above some threshold.

The *unwrap* keyword only applies to the dump *dcd* and *xtc* styles. If set to *yes*, coordinates will be written "unwrapped" by the image flags for each atom. Unwrapped means that if the atom has passed thru a periodic boundary one or more times, the value is printed for what the coordinate would be if it had not been wrapped back into the periodic box. Note that these coordinates may thus be far outside the box size stored with the snapshot.

These keywords apply only to the dump image and dump movie styles. Any keyword that affects an image, also affects a movie, since the movie is simply a collection of images. Some of the keywords only affect the

The *acolor* keyword can be used with the [dump image](#) command, when its atom color setting is *type*, to set the color that atoms of each type will be drawn in the image.

The specified *type* should be an integer from 1 to Ntypes = the number of atom types. A wildcard asterisk can be used in place of or in conjunction with the *type* argument to specify a range of atom types. This takes the form "*" or "*n" or "n*" or "m*n". If N = the number of atom types, then an asterisk with no numeric values means all types from 1 to N. A leading asterisk means all types from 1 to n (inclusive). A trailing asterisk means all types from n to N (inclusive). A middle asterisk means all types from m to n (inclusive).

The specified *color* can be a single color which is any of the 140 pre-defined colors (see below) or a color name defined by the dump_modify color option. Or it can be two or more colors separated by a "/" character, e.g. red/green/blue. In the former case, that color is assigned to all the specified atom types. In the latter case, the list of colors are assigned in a round-robin fashion to each of the specified atom types.

The *adiam* keyword can be used with the [dump image](#) command, when its atom diameter setting is *type*, to set the size that atoms of each type will be drawn in the image. The specified *type* should be an integer from 1 to Ntypes. As with the *acolor* keyword, a wildcard asterisk can be used as part of the *type* argument to specify a range of atomt types. The specified *diam* is the size in whatever distance [units](#) the input script is using, e.g. Angstroms.

The *amap* keyword can be used with the [dump image](#) command, with its *atom* keyword, when its atom setting is an atom-attribute, to setup a color map. The color map is used to assign a specific RGB (red/green/blue) color value to an individual atom when it is drawn, based on the atom's attribute, which is a numeric value, e.g. its x-component of velocity if the atom-attribute "vx" was specified.

The basic idea of a color map is that the atom-attribute will be within a range of values, and that range is associated with a a series of colors (e.g. red, blue, green). An atom's specific value (vx = -3.2) can then mapped to the series of colors (e.g. halfway between red and blue), and a specific color is determined via an interpolation procedure.

There are many possible options for the color map, enabled by the *amap* keyword. Here are the details.

The *lo* and *hi* settings determine the range of values allowed for the atom attribute. If numeric values are used for *lo* and/or *hi*, then values that are lower/higher than that value are set to the value. I.e. the range is static. If *lo* is specified as *min* or *hi* as *max* then the range is dynamic, and the lower and/or upper bound will be calculated each time an image is drawn, based on the set of atoms being visualized.

The *style* setting is two letters, such as "ca". The first letter is either "c" for continuous, "d" for discrete, or "s" for sequential. The second letter is either "a" for absolute, or "f" for fractional.

A continuous color map is one in which the color changes continuously from value to value within the range. A discrete color map is one in which discrete colors are assigned to sub-ranges of values within the range. A sequential color map is one in which discrete colors are assigned to a sequence of sub-ranges of values covering the entire range.

An absolute color map is one in which the values to which colors are assigned are specified explicitly as values within the range. A fractional color map is one in which the values to which colors are assigned are specified as a fractional portion of the range. For example if the range is from -10.0 to 10.0, and the color red is to be assigned to atoms with a value of 5.0, then for an absolute color map the number 5.0 would be used. But for a fractional map, the number 0.75 would be used since 5.0 is 3/4 of the way from -10.0 to 10.0.

The *delta* setting must be specified for all styles, but is only used for the sequential style; otherwise the value is ignored. It specifies the bin size to use within the range for assigning consecutive colors to. For example, if the range is from -10.0 to 10.0 and a *delta* of 1.0 is used, then 20 colors will be assigned to the range. The first will be from -10.0 <= color1 < -9.0, then 2nd from -9.0 <= color2 < -8.0, etc.

The *N* setting is how many entries follow. The format of the entries depends on whether the color map style is continuous, discrete or sequential. In all cases the *color* setting can be any of the 140 pre-defined colors (see below) or a color name defined by the dump_modify color option.

For continuous color maps, each entry has a *value* and a *color*. The *value* is either a number within the range of values or *min* or *max*. The *value* of the first entry must be *min* and the *value* of the last entry must be *max*. Any entries in between must have increasing values. Note that numeric values can be specified either as absolute numbers or as fractions (0.0 to 1.0) of the range, depending on the "a" or "f" in the style setting for the color map.

Here is how the entries are used to determine the color of an individual atom, given the value X of its atom attribute. X will fall between 2 of the entry values. The color of the atom is linearly interpolated (in each of the RGB values) between the 2 colors associated with those entries. For example, if X = -5.0 and the 2 surrounding entries are "red" at -10.0 and "blue" at 0.0, then the atom's color will be halfway between "red" and "blue", which happens to be "purple".

For discrete color maps, each entry has a *lo* and *hi* value and a *color*. The *lo* and *hi* settings are either numbers within the range of values or *lo* can be *min* or *hi* can be *max*. The *lo* and *hi* settings of the last entry must be *min* and *max*. Other entries can have any *lo* and *hi* values and the sub-ranges of different values can overlap. Note that numeric *lo* and *hi* values can be specified either as absolute numbers or as fractions (0.0 to 1.0) of the range, depending on the "a" or "f" in the style setting for the color map.

Here is how the entries are used to determine the color of an individual atom, given the value X of its atom attribute. The entries are scanned from first to last. The first time that *lo* <= X <= *hi*, X is assigned the color associated with that entry. You can think of the last entry as assigning a default color (since it will always be matched by X), and the earlier entries as colors that override the default. Also note that no interpolation of a color RGB is done. All atoms will be drawn with one of the colors in the list of entries.

For sequential color maps, each entry has only a *color*. Here is how the entries are used to determine the color of an individual atom, given the value X of its atom attribute. The range is partitioned into N bins of width *binsize*. Thus X will fall in a specific bin from 1 to N, say the Mth bin. If it falls on a boundary between 2 bins, it is considered to be in the higher of the 2 bins. Each bin is assigned a color from the E entries. If E < N, then the colors are repeated. For example if 2 entries with colors red and green are specified, then the odd numbered bins will be red and the even bins green. The color of the atom is the color of its bin. Note that the sequential color map is really a shorthand way of defining a discrete color map without having to specify where all the bin boundaries are.

The *backcolor* sets the background color of the images. The color name can be any of the 140 pre-defined colors (see below) or a color name defined by the dump_modify color option.

The *bcolor* keyword can be used with the dump image command, with its *bond* keyword, when its color setting is *type*, to set the color that bonds of each type will be drawn in the image.

The specified *type* should be an integer from 1 to Nbondtypes = the number of bond types. A wildcard asterisk can be used in place of or in conjunction with the *type* argument to specify a range of bond types. This takes the form "*" or "*n" or "n*" or "m*n". If N = the number of bond types, then an asterisk with no numeric values means all types from 1 to N. A leading asterisk means all types from 1 to n (inclusive). A trailing asterisk means all types from n to N (inclusive). A middle asterisk means all types from m to n (inclusive).

The specified *color* can be a single color which is any of the 140 pre-defined colors (see below) or a color name defined by the dump_modify color option. Or it can be two or more colors separated by a "/" character, e.g. red/green/blue. In the former case, that color is assigned to all the specified bond types. In the latter case, the list of colors are assigned in a round-robin fashion to each of the specified bond types.

The *bdiam* keyword can be used with the [dump image](#) command, with its *bond* keyword, when its diam setting is *type*, to set the diameter that bonds of each type will be drawn in the image. The specified *type* should be an integer from 1 to Nbondtypes. As with the *bcolor* keyword, a wildcard asterisk can be used as part of the *type* argument to specify a range of bond types. The specified *diam* is the size in whatever distance [units](#) you are using, e.g. Angstroms.

The *bitrate* keyword can be used with the [dump movie](#) command to define the size of the resulting movie file and its quality via setting how many kbits per second are to be used for the movie file. Higher bitrates require less compression and will result in higher quality movies. The quality is also determined by the compression format and encoder. The default setting is 2000 kbit/s, which will result in average quality with older compression formats.

IMPORTANT NOTE: Not all movie file formats supported by dump movie allow the bitrate to be set. If not, the setting is silently ignored.

The *boxcolor* keyword sets the color of the simulation box drawn around the atoms in each image. See the "dump image box" command for how to specify that a box be drawn. The color name can be any of the 140 pre-defined colors (see below) or a color name defined by the dump_modify color option.

The *color* keyword allows definition of a new color name, in addition to the 140-predefined colors (see below), and associates 3 red/green/blue RGB values with that color name. The color name can then be used with any other dump_modify keyword that takes a color name as a value. The RGB values should each be floating point values between 0.0 and 1.0 inclusive.

When a color name is converted to RGB values, the user-defined color names are searched first, then the 140 pre-defined color names. This means you can also use the *color* keyword to overwrite one of the pre-defined color names with new RBG values.

The *framerate* keyword can be used with the [dump movie](#) command to define the duration of the resulting movie file. Movie files written by the dump *movie* command have a default frame rate of 24 frames per second and the images generated will be converted at that rate. Thus a sequence of 1000 dump images will result in a movie of about 42 seconds. To make a movie run longer you can either generate images more frequently or lower the frame rate. To speed a movie up, you can do the inverse. Using a frame rate higher than 24 is not recommended, as it will result in simply dropping the rendered images. It is more efficient to dump images less frequently.

Restrictions: none

Related commands:

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

Default:

The option defaults are

- append = no

- buffer = yes for dump styles *atom*, *custom*, *loca*, and *xyz*
- element = "C" for every atom type
- every = whatever it was set to via the <u>dump</u> command
- fileper = # of processors
- first = no
- flush = yes
- format = %d and %g for each integer or floating point value
- image = no
- label = ENTRIES
- nfile = 1
- pad = 0
- precision = 1000
- region = none
- scale = yes
- sort = off for dump styles *atom*, *custom*, *cfg*, and *local*
- sort = id for dump styles *dcd*, *xtc*, and *xyz*
- thresh = none
- unwrap = no

- acolor = * red/green/blue/yellow/aqua/cyan
- adiam = * 1.0
- amap = min max cf 0.0 2 min blue max red
- backcolor = black
- bcolor = * red/green/blue/yellow/aqua/cyan
- bdiam = * 0.5
- bitrate = 2000
- boxcolor = yellow
- color = 140 color names are pre-defined as listed below
- framerate = 24

These are the standard 109 element names that LIGGGHTS(R)-WSU pre-defines for use with the <u>dump image</u> and dump_modify commands.

- 1-10 = "H", "He", "Li", "Be", "B", "C", "N", "O", "F", "Ne"
- 11-20 = "Na", "Mg", "Al", "Si", "P", "S", "Cl", "Ar", "K", "Ca"
- 21-30 = "Sc", "Ti", "V", "Cr", "Mn", "Fe", "Co", "Ni", "Cu", "Zn"
- 31-40 = "Ga", "Ge", "As", "Se", "Br", "Kr", "Rb", "Sr", "Y", "Zr"
- 41-50 = "Nb", "Mo", "Tc", "Ru", "Rh", "Pd", "Ag", "Cd", "In", "Sn"
- 51-60 = "Sb", "Te", "I", "Xe", "Cs", "Ba", "La", "Ce", "Pr", "Nd"
- 61-70 = "Pm", "Sm", "Eu", "Gd", "Tb", "Dy", "Ho", "Er", "Tm", "Yb"
- 71-80 = "Lu", "Hf", "Ta", "W", "Re", "Os", "Ir", "Pt", "Au", "Hg"
- 81-90 = "Tl", "Pb", "Bi", "Po", "At", "Rn", "Fr", "Ra", "Ac", "Th"
- 91-100 = "Pa", "U", "Np", "Pu", "Am", "Cm", "Bk", "Cf", "Es", "Fm"
- 101-109 = "Md", "No", "Lr", "Rf", "Db", "Sg", "Bh", "Hs", "Mt"

These are the 140 colors that LIGGGHTS(R)-WSU pre-defines for use with the <u>dump image</u> and dump_modify commands. Additional colors can be defined with the dump_modify color command. The 3 numbers listed for each name are the RGB (red/green/blue) values. Divide each value by 255 to get the equivalent 0.0 to 1.0 value.

aliceblue = 240, 248, 255	antiquewhite = 250, 235, 215	aqua = 0, 255, 255	aquamarine = 127, 255, 212	azure = 240, 255, 255
beige = 245, 245, 220	bisque = 255, 228, 196	black = 0, 0, 0	blanchedalmond = 255, 255, 205	blue = 0, 0, 255

blueviolet = 138, 43, 226	brown = 165, 42, 42	burlywood = 222, 184, 135	cadetblue = 95, 158, 160	chartreuse = 127, 255, 0
chocolate = 210, 105, 30	coral = 255, 127, 80	cornflowerblue = 100, 149, 237	cornsilk = 255, 248, 220	crimson = 220, 20, 60
cyan = 0, 255, 255	darkblue = 0, 0, 139	darkcyan = 0, 139, 139	darkgoldenrod = 184, 134, 11	darkgray = 169, 169, 169
darkgreen = 0, 100, 0	darkkhaki = 189, 183, 107	darkmagenta = 139, 0, 139	darkolivegreen = 85, 107, 47	darkorange = 255, 140, 0
darkorchid = 153, 50, 204	darkred = 139, 0, 0	darksalmon = 233, 150, 122	darkseagreen = 143, 188, 143	darkslateblue = 72, 61, 139
darkslategray = 47, 79, 79	darkturquoise = 0, 206, 209	darkviolet = 148, 0, 211	deeppink = 255, 20, 147	deepskyblue = 0, 191, 255
dimgray = 105, 105, 105	dodgerblue = 30, 144, 255	firebrick = 178, 34, 34	floralwhite = 255, 250, 240	forestgreen = 34, 139, 34
fuchsia = 255, 0, 255	gainsboro = 220, 220, 220	ghostwhite = 248, 248, 255	gold = 255, 215, 0	goldenrod = 218, 165, 32
gray = 128, 128, 128	green = 0, 128, 0	greenyellow = 173, 255, 47	honeydew = 240, 255, 240	hotpink = 255, 105, 180
indianred = 205, 92, 92	indigo = 75, 0, 130	ivory = 255, 240, 240	khaki = 240, 230, 140	lavender = 230, 230, 250
lavenderblush = 255, 240, 245	lawngreen = 124, 252, 0	lemonchiffon = 255, 250, 205	lightblue = 173, 216, 230	lightcoral = 240, 128, 128
lightcyan = 224, 255, 255	lightgoldenrodyellow = 250, 250, 210	lightgreen = 144, 238, 144	lightgrey = 211, 211, 211	lightpink = 255, 182, 193
lightsalmon = 255, 160, 122	lightseagreen = 32, 178, 170	lightskyblue = 135, 206, 250	lightslategray = 119, 136, 153	lightsteelblue = 176, 196, 222
lightyellow = 255, 255, 224	lime = 0, 255, 0	limegreen = 50, 205, 50	linen = 250, 240, 230	magenta = 255, 0, 255
maroon = 128, 0, 0	mediumaquamarine = 102, 205, 170	mediumblue = 0, 0, 205	mediumorchid = 186, 85, 211	mediumpurple = 147, 112, 219
mediumseagreen = 60, 179, 113	mediumslateblue = 123, 104, 238	mediumspringgreen = 0, 250, 154	mediumturquoise = 72, 209, 204	mediumvioletred = 199, 21, 133
midnightblue = 25, 25, 112	mintcream = 245, 255, 250	mistyrose = 255, 228, 225	moccasin = 255, 228, 181	navajowhite = 255, 222, 173
navy = 0, 0, 128	oldlace = 253, 245, 230	olive = 128, 128, 0	olivedrab = 107, 142, 35	orange = 255, 165, 0
orangered = 255, 69, 0	orchid = 218, 112, 214	palegoldenrod = 238, 232, 170	palegreen = 152, 251, 152	paleturquoise = 175, 238, 238
palevioletred = 219, 112, 147	papayawhip = 255, 239, 213	peachpuff = 255, 239, 213	peru = 205, 133, 63	pink = 255, 192, 203
plum = 221, 160, 221	powderblue = 176, 224, 230	purple = 128, 0, 128	red = 255, 0, 0	rosybrown = 188, 143, 143
royalblue = 65, 105, 225	saddlebrown = 139, 69, 19	salmon = 250, 128, 114	sandybrown = 244, 164, 96	seagreen = 46, 139, 87
seashell = 255, 245, 238	sienna = 160, 82, 45	silver = 192, 192, 192	skyblue = 135, 206, 235	slateblue = 106, 90, 205
slategray = 112, 128, 144	snow = 255, 250, 250	springgreen = 0, 255, 127	steelblue = 70, 130, 180	tan = 210, 180, 140
teal = 0, 128, 128	thistle = 216, 191, 216	tomato = 253, 99, 71		

			turquoise = 64, 224, 208	violet = 238, 130, 238
wheat = 245, 222, 179	white = 255, 255, 255	whitesmoke = 245, 245, 245	yellow = 255, 255, 0	yellowgreen = 154, 205, 50

dump_modify command

Syntax:

```
dump_modify dump-ID keyword values ...
```

- dump-ID = ID of dump to modify
- one or more keyword/value pairs may be appended
- these keywords apply to the dump custom/vtk style
- keyword = *binary* or *element* or *every* or *fileper* or *first* or *label* or *nfile* or *pad* or *region* or *sort* or *thresh*

```
binary arg = yes or no
  element args = E1 E2 ... EN, where N = # of atom types
    E1,...,EN = element name, e.g. C or Fe or Ga
  every arg = N
    N = dump every this many timesteps
    N can be a variable (see below)
  fileper arg = Np
    Np = write one file for every this many processors
  first arg = yes or no
  label arg = string
    string = character string to use in header of legacy VTK file
  nfile arg = Nf
    Nf = write this many files, one from each of Nf processors
  pad arg = Nchar = # of characters to convert timestep to
  region arg = region-ID or "none"
  sort arg = off or id or N or -N
     off = no sorting of per-atom lines within a snapshot
     id = sort per-atom lines by atom ID
     N = sort per-atom lines in ascending order by the Nth column
     -N = sort per-atom lines in descending order by the Nth column
  thresh args = attribute operation value
    attribute = same attributes (x,fy,etotal,sxx,etc) used by dump custom style
    operation = "" or ">=" or "==" or "!="
    value = numeric value to compare to
    these 3 args can be replaced by the word "none" to turn off thresholding
```

Examples:

```
dump_modify dmpvtp binary yes
dump_modify e_data region mySphere thresh x <0.0 thresh ervel >= 0.2
dump_modify dmpvtk every 1000 nfile 20
dump_modify dmpvtk every v_myVar
```

Description:

Modify the parameters of a previously defined dump command.

These keywords apply to the [dump custom/vtk](#) style. The description gives details.

The *binary* keyword, if specified as *yes*, causes the output to be written in binary format. If specified as *no*, which is the default, the data is written in ASCII format to the output file.

The *element* keyword associates element names (e.g. H, C, Fe) with LIGGGHTS(R)-WSU atom types. There are no restrictions to what label can be used as an element name. Any whitespace separated text will be

accepted.

The *every* keyword changes the dump frequency originally specified by the dump custom/vtk command to a new value. The every keyword can be specified in one of two ways. It can be a numeric value in which case it must be > 0. Or it can be an equal-style variable, which should be specified as v_name, where name is the variable name.

In this case, the variable is evaluated at the beginning of a run to determine the next timestep at which a dump snapshot will be written out. On that timestep the variable will be evaluated again to determine the next timestep, etc. Thus the variable should return timestep values. See the stagger() and logfreq() and stride() math functions for equal-style variables, as examples of useful functions to use in this context. Other similar math functions could easily be added as options for equal-style variables. Also see the next() function, which allows use of a file-style variable which reads successive values from a file, each time the variable is evaluated. Used with the *every* keyword, if the file contains a list of ascending timesteps, you can output snapshots whenever you wish.

Note that when using the variable option with the *every* keyword, you need to use the *first* option if you want an initial snapshot written to the dump file.

For example, the following commands will write snapshots at timesteps 0,10,20,30,100,200,300,1000,2000,etc:

```
variable        s equal logfreq(10,3,10)
dump            1 all custom/vtk 100 tmp.dump*.vtk vx vy vz
dump_modify     1 every v_s first yes
```

The following commands would write snapshots at the timesteps listed in file tmp.times:

```
variable        f file tmp.times
variable        s equal next(f)
dump            1 all custom/vtk 100 tmp.dump*.vtk vx vy vz
dump_modify     1 every v_s
```

IMPORTANT NOTE: When using a file-style variable with the *every* keyword, the file of timesteps must list a first timestep that is beyond the current timestep (e.g. it cannot be 0). And it must list one or more timesteps beyond the length of the run you perform. This is because the dump command will generate an error if the next timestep it reads from the file is not a value greater than the current timestep. Thus if you wanted output on steps 0,15,100 of a 100-timestep run, the file should contain the values 15,100,101 and you should also use the dump_modify first command. Any final value > 100 could be used in place of 101.

The *first* keyword determines whether a dump snapshot is written on the very first timestep after the dump command is invoked. This will always occur if the current timestep is a multiple of N, the frequency specified in the dump custom/vtk command, including timestep 0. But if this is not the case, a dump snapshot will only be written if the setting of this keyword is *yes*. If it is *no*, which is the default, then it will not be written.

The *fileper* keyword is documented below with the *nfile* keyword.

When writing to legacy VTK files, the dump *custom/vtk* style will use the specified *label* as the header line. By default this header line is:

```
Generated by LIGGGHTS(R)-WSU
```

The *nfile* or *fileper* keywords can be used in conjunction with the "%" wildcard character in the specified dump file name, if an XML file format was specified. As explained on the dump custom_vtk command doc page, the "%" character causes the dump file to be written in pieces, one piece for each of P processors. By

default P = the number of processors the simulation is running on. The *nfile* or *fileper* keyword can be used to set P to a smaller value, which can be more efficient when running on a large number of processors.

The *nfile* keyword sets P to the specified Nf value. For example, if Nf = 4, and the simulation is running on 100 processors, 4 files will be written, by processors 0,25,50,75. Each will collect information from itself and the next 24 processors and write it to a dump file.

For the *fileper* keyword, the specified value of Np means write one file for every Np processors. For example, if Np = 4, every 4th processor (0,4,8,12,etc) will collect information from itself and the next 3 processors and write it to a dump file.

The *pad* keyword only applies when the dump filename is specified with a wildcard "*" character which becomes the timestep. If *pad* is 0, which is the default, the timestep is converted into a string of unpadded length, e.g. 100 or 12000 or 2000000. When *pad* is specified with *Nchar* > 0, the string is padded with leading zeroes so they are all the same length = *Nchar*. For example, pad 7 would yield 0000100, 0012000, 2000000. This can be useful so that post-processing programs can easily read the files in ascending timestep order.

If the *region* keyword is specified, only atoms in the region will be written to the dump file. Only one region can be applied as a filter (the last one specified). See the region command for more details. Note that a region can be defined as the "inside" or "outside" of a geometric shape, and it can be the "union" or "intersection" of a series of simpler regions.

The *sort* keyword determines whether lines of per-atom output in a snapshot are sorted or not. A sort value of *off* means they will typically be written in indeterminate order, either in serial or parallel. This is the case even in serial if the atom_modify sort option is turned on, which it is by default, to improve performance. A sort value of *id* means sort the output by atom ID. A sort value of N or -N means sort the output by the value in the Nth column of per-atom info in either ascending or descending order.

IMPORTANT NOTE: Unless it is required by the dump style, sorting dump file output requires extra overhead in terms of CPU and communication cost, as well as memory, versus unsorted output.

Using the *thresh* keyword, multiple thresholds can be specified. Specifying "none" turns off all threshold criteria. If thresholds are specified, only atoms whose attributes meet all the threshold criteria are written to the dump file. The possible attributes that can be tested for are the same as those that can be specified in the dump custom/vtk command, with the exception of the *element* attribute, since it is not a numeric value. Note that different attributes can be output by the dump custom/vtk command than are used as threshold criteria by the dump_modify command. E.g. you can output the coordinates and stress of atoms whose energy is above some threshold.

Restrictions: none

Related commands:

dump, dump custom/vtk, dump image, undump

Default:

The option defaults are

- binary = no
- element = "C" for every atom type
- every = whatever it was set to via the dump custom/vtk command
- fileper = # of processors

dump_modify command   143

- first = no
- label = "Generated by LIGGGHTS"
- nfile = 1
- pad = 0
- region = none
- sort = off
- thresh = none

- first = no
- label = "Generated by LIGGGHTS"

- nfile = 1

- pad = 0

echo command

Syntax:

```
echo style
```

- style = *none* or *screen* or *log* or *both*

Examples:

```
echo both
echo log
```

Description:

This command determines whether LIGGGHTS(R)-WSU echoes each input script command to the screen and/or log file as it is read and processed. If an input script has errors, it can be useful to look at echoed output to see the last command processed.

The command-line switch -echo can be used in place of this command.

Restrictions: none

Related commands: none

Default:

```
echo log
```

fix adapt command

Syntax:

```
fix ID group-ID adapt N attribute args ... keyword value ...
```

- ID, group-ID are documented in fix command
- adapt = style name of this fix command
- N = adapt simulation settings every this many timesteps
- one or more attribute/arg pairs may be appended
- attribute = *pair* or *atom*

```
  pair args = pstyle pparam I J v_name
      pstyle = pair style name, e.g. soft
      pparam = parameter to adapt over time
      I,J = type pair(s) to set parameter for
      v_name = variable with name that calculates value of pparam
    atom args = aparam v_name
      aparam = parameter to adapt over time
      v_name = variable with name that calculates value of aparam
```

- zero or more keyword/value pairs may be appended
- keyword = *scale* or *reset*

```
  scale value = no or yes
      no = the variable value is the new setting
      yes = the variable value multiplies the original setting
    reset value = no or yes
      no = values will remain altered at the end of a run
      yes = reset altered values to their original values at the end of a run
```

Examples:

```
fix 1 all adapt 1 pair soft a 1 1 v_prefactor
fix 1 all adapt 1 pair soft a 2* 3 v_prefactor
fix 1 all adapt 10 atom diameter v_size
```

Description:

Change or adapt one or more specific simulation attributes or settings over time as a simulation runs. Pair potential and K-space and atom attributes which can be varied by this fix are discussed below. Many other fixes can also be used to time-vary simulation parameters, e.g. the "fix deform" command will change the simulation box size/shape and the "fix move" command will change atom positions and velocities in a prescribed manner. Also note that many commands allow variables as arguments for specific parameters, if described in that manner on their doc pages. An equal-style variable can calculate a time-dependent quantity, so this is another way to vary a simulation parameter over time.

If *N* is specified as 0, the specified attributes are only changed once, before the simulation begins. This is all that is needed if the associated variables are not time-dependent. If *N* > 0, then changes are made every *N* steps during the simulation, presumably with a variable that is time-dependent.

Depending on the value of the *reset* keyword, attributes changed by this fix will or will not be reset back to their original values at the end of a simulation. Even if *reset* is specified as *yes*, a restart file written during a simulation will contain the modified settings.

If the *scale* keyword is set to *no*, then the value the parameter is set to will be whatever the variable generates. If the *scale* keyword is set to *yes*, then the value of the altered parameter will be the initial value of that parameter multiplied by whatever the variable generates. I.e. the variable is now a "scale factor" applied in (presumably) a time-varying fashion to the parameter. Internally, the parameters themselves are actually altered; make sure you use the *reset yes* option if you want the parameters to be restored to their initial values after the run.

The *pair* keyword enables various parameters of potentials defined by the pair_style command to be changed, if the pair style supports it. Note that the pair_style and pair_coeff commands must be used in the usual manner to specify these parameters initially; the fix adapt command simply overrides the parameters.

The *pstyle* argument is the name of the pair style. If pair_style hybrid or hybrid/overlay is used, *pstyle* should be a sub-style name. For example, *pstyle* could be specified as "soft" or "lubricate". The *pparam* argument is the name of the parameter to change. This is the current list of pair styles and parameters that can be varied by this fix. See the doc pages for individual pair styles and their energy formulas for the meaning of these parameters:

soft	a	type pairs

IMPORTANT NOTE: It is easy to add new potentials and their parameters to this list. All it typically takes is adding an extract() method to the pair_*.cpp file associated with the potential.

Some parameters are global settings for the pair style, e.g. the viscosity setting "mu" for pair_style lubricate. Other parameters apply to atom type pairs within the pair style, e.g. the prefactor "a" for pair_style soft.

If a type pair parameter is specified, the *I* and *J* settings should be specified to indicate which type pairs to apply it to. If a global parameter is specified, the *I* and *J* settings still need to be specified, but are ignored.

Similar to the pair_coeff command, I and J can be specified in one of two ways. Explicit numeric values can be used for each, as in the 1st example above. I <= J is required. LIGGGHTS(R)-WSU sets the coefficients for the symmetric J,I interaction to the same values.

A wild-card asterisk can be used in place of or in conjunction with the I,J arguments to set the coefficients for multiple pairs of atom types. This takes the form "*" or "*n" or "n*" or "m*n". If N = the number of atom types, then an asterisk with no numeric values means all types from 1 to N. A leading asterisk means all types from 1 to n (inclusive). A trailing asterisk means all types from n to N (inclusive). A middle asterisk means all types from m to n (inclusive). Note that only type pairs with I <= J are considered; if asterisks imply type pairs where J < I, they are ignored.

IMPROTANT NOTE: If pair_style hybrid or hybrid/overlay is being used, then the *pstyle* will be a sub-style name. You must specify I,J arguments that correspond to type pair values defined (via the pair_coeff command) for that sub-style.

The *v_name* argument for keyword *pair* is the name of an equal-style variable which will be evaluated each time this fix is invoked to set the parameter to a new value. It should be specified as v_name, where name is the variable name. Equal-style variables can specify formulas with various mathematical functions, and include thermo_style command keywords for the simulation box parameters and timestep and elapsed time. Thus it is easy to specify parameters that change as a function of time or span consecutive runs in a continuous fashion. For the latter, see the *start* and *stop* keywords of the run command and the *elaplong* keyword of thermo_style custom for details.

For example, these commands would change the prefactor coefficient of the pair_style soft potential from 10.0 to 30.0 in a linear fashion over the course of a simulation:

```
variable prefactor equal ramp(10,30)
fix 1 all adapt 1 pair soft a * * v_prefactor
```

The *atom* keyword enables various atom properties to be changed. The *aparam* argument is the name of the parameter to change. This is the current list of atom parameters that can be varied by this fix:

- charge = charge on particle
- diameter = diameter of particle

The *v_name* argument of the *atom* keyword is the name of an [equal-style variable](#) which will be evaluated each time this fix is invoked to set the parameter to a new value. It should be specified as v_name, where name is the variable name. See the discussion above describing the formulas associated with equal-style variables. The new value is assigned to the corresponding attribute for all atoms in the fix group.

If the atom parameter is *diameter* and per-atom density and per-atom mass are defined for particles (e.g. [atom_style granular](#)), then the mass of each particle is also changed when the diameter changes (density is assumed to stay constant).

For example, these commands would shrink the diameter of all granular particles in the "center" group from 1.0 to 0.1 in a linear fashion over the course of a 1000-step simulation:

```
variable size equal ramp(1.0,0.1)
fix 1 center adapt 10 atom diameter v_size
```

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

No information about this fix is written to [binary restart files](#). None of the [fix_modify](#) options are relevant to this fix. No global or per-atom quantities are stored by this fix for access by various [output commands](#). No parameter of this fix can be used with the *start/stop* keywords of the [run](#) command. This fix is not invoked during [energy minimization](#).

Restrictions: none

Related commands:

[compute ti](#)

Default:

The option defaults are scale = no, reset = no.

fix addforce command

Syntax:

```
fix ID group-ID addforce fx fy fz keyword value ...
```

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

```
    any of fx,fy,fz can be a variable (see below)
```
- zero or more keyword/value pairs may be appended to args
- keyword = *region* or *energy*

```
  region value = region-ID
     region-ID = ID of region atoms must be in to have added force
   energy value = v_name
      v_name = variable with name that calculates the potential energy of each atom in the a
```

Examples:

```
fix kick flow addforce 1.0 0.0 0.0
fix kick flow addforce 1.0 0.0 v_oscillate
fix ff boundary addforce 0.0 0.0 v_push energy v_espace
```

Description:

Add fx,fy,fz to the corresponding component of force for each atom in the group. This command can be used to give an additional push to atoms in a simulation, such as for a simulation of Poiseuille flow in a channel.

Any of the 3 quantities defining the force components can be specified as an equal-style or atom-style variable, namely *fx*, *fy*, *fz*. If the value is a variable, it should be specified as v_name, where name is the variable name. In this case, the variable will be evaluated each timestep, and its value(s) used to determine the force component.

Equal-style variables can specify formulas with various mathematical functions, and include thermo_style command keywords for the simulation box parameters and timestep and elapsed time. Thus it is easy to specify a time-dependent force field.

Atom-style variables can specify the same formulas as equal-style variables but can also include per-atom values, such as atom coordinates. Thus it is easy to specify a spatially-dependent force field with optional time-dependence as well.

If the *region* keyword is used, the atom must also be in the specified geometric region in order to have force added to it.

Adding a force to atoms implies a change in their potential energy as they move due to the applied force field. For dynamics via the "run" command, this energy can be optionally added to the system's potential energy for thermodynamic output (see below). For energy minimization via the "minimize" command, this energy must be added to the system's potential energy to formulate a self-consistent minimization problem (see below).

The *energy* keyword is not allowed if the added force is a constant vector F = (fx,fy,fz), with all components defined as numeric constants and not as variables. This is because LIGGGHTS(R)-WSU can compute the

fix addforce command   149

energy for each atom directly as E = -x dot F = -(x*fx + y*fy + z*fz), so that -Grad(E) = F.

The *energy* keyword is optional if the added force is defined with one or more variables, and if you are performing dynamics via the [run](run) command. If the keyword is not used, LIGGGHTS(R)-WSU will set the energy to 0.0, which is typically fine for dynamics.

The *energy* keyword is required if the added force is defined with one or more variables, and you are performing energy minimization via the "minimize" command. The keyword specifies the name of an atom-style variable which is used to compute the energy of each atom as function of its position. Like variables used for *fx*, *fy*, *fz*, the energy variable is specified as v_name, where name is the variable name.

Note that when the *energy* keyword is used during an energy minimization, you must insure that the formula defined for the atom-style variable is consistent with the force variable formulas, i.e. that -Grad(E) = F. For example, if the force were a spring-like F = kx, then the energy formula should be E = -0.5kx^2. If you don't do this correctly, the minimization will not converge properly.

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

No information about this fix is written to binary restart files.

The fix_modify *energy* option is supported by this fix to add the potential "energy" inferred by the added force to the system's potential energy as part of thermodynamic output. This is a fictitious quantity but is needed so that the minimize command can include the forces added by this fix in a consistent manner. I.e. there is a decrease in potential energy when atoms move in the direction of the added force.

This fix computes a global scalar and a global 3-vector of forces, which can be accessed by various output commands. The scalar is the potential energy discussed above. The vector is the total force on the group of atoms before the forces on individual atoms are changed by the fix. The scalar and vector values calculated by this fix are "extensive".

No parameter of this fix can be used with the *start/stop* keywords of the [run](run) command.

The forces due to this fix are imposed during an energy minimization, invoked by the minimize command. You should not specify force components with a variable that has time-dependence for use with a minimizer, since the minimizer increments the timestep as the iteration count during the minimization.

IMPORTANT NOTE: If you want the fictitious potential energy associated with the added forces to be included in the total potential energy of the system (the quantity being minimized), you MUST enable the fix_modify *energy* option for this fix.

Restrictions: none

Related commands:

fix setforce, fix aveforce

Default: none

fix ave/atom command

Syntax:

```
fix ID group-ID ave/atom Nevery Nrepeat Nfreq value1 value2 ...
```

- ID, group-ID are documented in fix command
- ave/atom = style name of this fix command
- Nevery = use input values every this many timesteps
- Nrepeat = # of times to use input values for calculating averages
- Nfreq = calculate averages every this many timesteps one or more input values can be listed
- value = x, y, z, vx, vy, vz, fx, fy, fz, c_ID, c_ID[i], f_ID, f_ID[i], v_name

```
x,y,z,vx,vy,vz,fx,fy,fz = atom attribute (position, velocity, force component)
c_ID = per-atom vector calculated by a compute with ID
c_ID[I] = Ith column of per-atom array calculated by a compute with ID
f_ID = per-atom vector calculated by a fix with ID
f_ID[I] = Ith column of per-atom array calculated by a fix with ID
v_name = per-atom vector calculated by an atom-style variable with name
```

Examples:

```
fix 1 all ave/atom 1 100 100 vx vy vz
fix 1 all ave/atom 10 20 1000 c_my_stress[1]
```

Description:

Use one or more per-atom vectors as inputs every few timesteps, and average them atom by atom over longer timescales. The resulting per-atom averages can be used by other output commands such as the fix ave/spatial or dump custom commands.

The group specified with the command means only atoms within the group have their averages computed. Results are set to 0.0 for atoms not in the group.

Each input value can be an atom attribute (position, velocity, force component) or can be the result of a compute or fix or the evaluation of an atom-style variable. In the latter cases, the compute, fix, or variable must produce a per-atom vector, not a global quantity or local quantity. If you wish to time-average global quantities from a compute, fix, or variable, then see the fix ave/time command.

Computes that produce per-atom vectors or arrays are those which have the word *atom* in their style name. See the doc pages for individual fixes to determine which ones produce per-atom vectors or arrays. Variables of style *atom* are the only ones that can be used with this fix since they produce per-atom vectors.

Each per-atom value of each input vector is averaged independently.

The *Nevery*, *Nrepeat*, and *Nfreq* arguments specify on what timesteps the input values will be used in order to contribute to the average. The final averaged quantities are generated on timesteps that are a multiple of *Nfreq*. The average is over *Nrepeat* quantities, computed in the preceding portion of the simulation every *Nevery* timesteps. *Nfreq* must be a multiple of *Nevery* and *Nevery* must be non-zero even if *Nrepeat* is 1. Also, the timesteps contributing to the average value cannot overlap, i.e. Nfreq > (Nrepeat-1)*Nevery is required.

For example, if Nevery=2, Nrepeat=6, and Nfreq=100, then values on timesteps 90,92,94,96,98,100 will be used to compute the final average on timestep 100. Similarly for timesteps 190,192,194,196,198,200 on

fix ave/atom command   151

timestep 200, etc.

The atom attribute values (x,y,z,vx,vy,vz,fx,fy,fz) are self-explanatory. Note that other atom attributes can be used as inputs to this fix by using the compute property/atom command and then specifying an input value from that compute.

IMPORTANT NOTE: The x,y,z attributes are values that are re-wrapped inside the periodic box whenever an atom crosses a periodic boundary. Thus if you time average an atom that spends half its time on either side of the periodic box, you will get a value in the middle of the box. If this is not what you want, consider averaging unwrapped coordinates, which can be provided by the compute property/atom command via its xu,yu,zu attributes.

If a value begins with "c_", a compute ID must follow which has been previously defined in the input script. If no bracketed term is appended, the per-atom vector calculated by the compute is used. If a bracketed term containing an index I is appended, the Ith column of the per-atom array calculated by the compute is used. Users can also write code for their own compute styles and add them to LIGGGHTS(R)-WSU.

If a value begins with "f_", a fix ID must follow which has been previously defined in the input script. If no bracketed term is appended, the per-atom vector calculated by the fix is used. If a bracketed term containing an index I is appended, the Ith column of the per-atom array calculated by the fix is used. Note that some fixes only produce their values on certain timesteps, which must be compatible with *Nevery*, else an error will result. Users can also write code for their own fix styles and add them to LIGGGHTS(R)-WSU.

If a value begins with "v_", a variable name must follow which has been previously defined in the input script as an atom-style variable Variables of style *atom* can reference thermodynamic keywords, or invoke other computes, fixes, or variables when they are evaluated, so this is a very general means of generating per-atom quantities to time average.

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

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix. No global scalar or vector quantities are stored by this fix for access by various output commands.

This fix produces a per-atom vector or array which can be accessed by various output commands. A vector is produced if only a single quantity is averaged by this fix. If two or more quantities are averaged, then an array of values is produced. The per-atom values can only be accessed on timesteps that are multiples of *Nfreq* since that is when averaging is performed.

No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions: none

Related commands:

compute, fix ave/histo, fix ave/spatial, fix ave/time, variable,

Default: none

fix ave/correlate command

Syntax:

```
fix ID group-ID ave/correlate Nevery Nrepeat Nfreq value1 value2 ... keyword args ...
```

- ID, group-ID are documented in fix command
- ave/correlate = style name of this fix command
- Nevery = use input values every this many timesteps
- Nrepeat = # of correlation time windows to accumulate
- Nfreq = calculate tine window averages every this many timesteps
- one or more input values can be listed
- value = c_ID, c_ID[N], f_ID, f_ID[N], v_name

```
    c_ID = global scalar calculated by a compute with ID
    c_ID[I] = Ith component of global vector calculated by a compute with ID
    f_ID = global scalar calculated by a fix with ID
    f_ID[I] = Ith component of global vector calculated by a fix with ID
    v_name = global value calculated by an equal-style variable with name
```

- zero or more keyword/arg pairs may be appended
- keyword = *type* or *ave* or *start* or *prefactor* or *file* or *overwrite* or *title1* or *title2* or *title3*

```
  type arg = auto or upper or lower or auto/upper or auto/lower or full
      auto = correlate each value with itself
      upper = correlate each value with each succeeding value
      lower = correlate each value with each preceding value
      auto/upper = auto + upper
      auto/lower = auto + lower
      full = correlate each value with every other value, including itself = auto + upper + 
  ave args = one or running
      one = zero the correlation accumulation every Nfreq steps
      running = accumulate correlations continuously
  start args = Nstart
      Nstart = start accumulating correlations on this timestep
  prefactor args = value
      value = prefactor to scale all the correlation data by
  file arg = filename
      filename = name of file to output correlation data to
  overwrite arg = none = overwrite output file with only latest output
  title1 arg = string
      string = text to print as 1st line of output file
  title2 arg = string
      string = text to print as 2nd line of output file
  title3 arg = string
      string = text to print as 3rd line of output file
```

Examples:

```
fix 1 all ave/correlate 5 100 1000 c_myTemp file temp.correlate
fix 1 all ave/correlate 1 50 10000 &
       c_thermo_press[1] c_thermo_press[2] c_thermo_press[3] &
       type upper ave running title1 "My correlation data"
```

Description:

Use one or more global scalar values as inputs every few timesteps, calculate time correlations bewteen them at varying time intervals, and average the correlation data over longer timescales. The resulting correlation

values can be time integrated by [variables](#) or used by other [output commands](#) such as [thermo_style custom](#), and can also be written to a file.

The group specified with this command is ignored. However, note that specified values may represent calculations performed by computes and fixes which store their own "group" definitions.

Each listed value can be the result of a [compute](#) or [fix](#) or the evaluation of an equal-style [variable](#). In each case, the compute, fix, or variable must produce a global quantity, not a per-atom or local quantity. If you wish to spatial- or time-average or histogram per-atom quantities from a compute, fix, or variable, then see the [fix ave/spatial](#), [fix ave/atom](#), or [fix ave/histo](#) commands. If you wish to sum a per-atom quantity into a single global quantity, see the [compute reduce](#) command.

[Computes](#) that produce global quantities are those which do not have the word *atom* in their style name. Only a few [fixes](#) produce global quantities. See the doc pages for individual fixes for info on which ones produce such values. [Variables](#) of style *equal* are the only ones that can be used with this fix. Variables of style *atom* cannot be used, since they produce per-atom values.

The input values must either be all scalars. What kinds of correlations between input values are calculated is determined by the *type* keyword as discussed below.

The *Nevery*, *Nrepeat*, and *Nfreq* arguments specify on what timesteps the input values will be used to calculate correlation data. The input values are sampled every *Nevery* timesteps. The correlation data for the preceding samples is computed on timesteps that are a multiple of *Nfreq*. Consider a set of samples from some initial time up to an output timestep. The initial time could be the beginning of the simulation or the last output time; see the *ave* keyword for options. For the set of samples, the correlation value Cij is calculated as:

```
Cij(delta) = ave(Vi(t)*Vj(t+delta))
```

which is the correlation value between input values Vi and Vj, separated by time delta. Note that the second value Vj in the pair is always the one sampled at the later time. The ave() represents an average over every pair of samples in the set that are separated by time delta. The maximum delta used is of size (*Nrepeat*-1)*Nevery*. Thus the correlation between a pair of input values yields *Nrepeat* correlation datums:

```
Cij(0), Cij(Nevery), Cij(2*Nevery), ..., Cij((Nrepeat-1)*Nevery)
```

For example, if Nevery=5, Nrepeat=6, and Nfreq=100, then values on timesteps 0,5,10,15,...,100 will be used to compute the final averages on timestep 100. Six averages will be computed: Cij(0), Cij(5), Cij(10), Cij(15), Cij(20), and Cij(25). Cij(10) on timestep 100 will be the average of 19 samples, namely Vi(0)*Vj(10), Vi(5)*Vj(15), Vi(10)*V j20), Vi(15)*Vj(25), ..., Vi(85)*Vj(95), Vi(90)*Vj(100).

Nfreq must be a multiple of *Nevery*; *Nevery* and *Nrepeat* must be non-zero. Also, if the *ave* keyword is set to *one* which is the default, then *Nfreq* >= (*Nrepeat*-1)*Nevery* is required.

If a value begins with "c_", a compute ID must follow which has been previously defined in the input script. If no bracketed term is appended, the global scalar calculated by the compute is used. If a bracketed term is appended, the Ith element of the global vector calculated by the compute is used.

Note that there is a [compute reduce](#) command which can sum per-atom quantities into a global scalar or vector which can thus be accessed by fix ave/correlate. Or it can be a compute defined not in your input script, but by [thermodynamic output](#) or other fixes such as [fix nvt](#) or [fix temp/rescale](#). See the doc pages for these commands which give the IDs of these computes. Users can also write code for their own compute styles and [add them to LIGGGHTS(R)-WSU](#).

If a value begins with "f_", a fix ID must follow which has been previously defined in the input script. If no bracketed term is appended, the global scalar calculated by the fix is used. If a bracketed term is appended, the

Ith element of the global vector calculated by the fix is used.

Note that some fixes only produce their values on certain timesteps, which must be compatible with *Nevery*, else an error will result. Users can also write code for their own fix styles and add them to LIGGGHTS(R)-WSU.

If a value begins with "v_", a variable name must follow which has been previously defined in the input script. Only equal-style variables can be referenced. See the variable command for details. Note that variables of style *equal* define a formula which can reference individual atom properties or thermodynamic keywords, or they can invoke other computes, fixes, or variables when they are evaluated, so this is a very general means of specifying quantities to time correlate.

Additional optional keywords also affect the operation of this fix.

The *type* keyword determines which pairs of input values are correlated with each other. For N input values Vi, for i = 1 to N, let the number of pairs = Npair. Note that the second value in the pair Vi(t)*Vj(t+delta) is always the one sampled at the later time.

- If *type* is set to *auto* then each input value is correlated with itself. I.e. Cii = Vi*Vi, for i = 1 to N, so Npair = N.
- If *type* is set to *upper* then each input value is correlated with every succeeding value. I.e. Cij = Vi*Vj, for i < j, so Npair = N*(N-1)/2.
- If *type* is set to *lower* then each input value is correlated with every preceeding value. I.e. Cij = Vi*Vj, for i > j, so Npair = N*(N-1)/2.
- If *type* is set to *auto/upper* then each input value is correlated with itself and every succeeding value. I.e. Cij = Vi*Vj, for i >= j, so Npair = N*(N+1)/2.
- If *type* is set to *auto/lower* then each input value is correlated with itself and every preceding value. I.e. Cij = Vi*Vj, for i <= j, so Npair = N*(N+1)/2.
- If *type* is set to *full* then each input value is correlated with itself and every other value. I.e. Cij = Vi*Vj, for i,j = 1,N so Npair = N^2.

The *ave* keyword determines what happens to the accumulation of correlation samples every *Nfreq* timesteps. If the *ave* setting is *one*, then the accumulation is restarted or zeroed every *Nfreq* timesteps. Thus the outputs on successive *Nfreq* timesteps are essentially independent of each other. The exception is that the Cij(0) = Vi(T)*Vj(T) value at a timestep T, where T is a multiple of *Nfreq*, contributes to the correlation output both at time T and at time T+Nfreq.

If the *ave* setting is *running*, then the accumulation is never zeroed. Thus the output of correlation data at any timestep is the average over samples accumulated every *Nevery* steps since the fix was defined. it can only be restarted by deleting the fix via the unfix command, or by re-defining the fix by re-specifying it.

The *start* keyword specifies what timestep the accumulation of correlation samples will begin on. The default is step 0. Setting it to a larger value can avoid adding non-equilibrated data to the correlation averages.

The *prefactor* keyword specifies a constant which will be used as a multiplier on the correlation data after it is averaged. It is effectively a scale factor on Vi*Vj, which can be used to account for the size of the time window or other unit conversions.

The *file* keyword allows a filename to be specified. Every *Nfreq* steps, an array of correlation data is written to the file. The number of rows is *Nrepeat*, as described above. The number of columns is the Npair+2, also as described above. Thus the file ends up to be a series of these array sections.

The *overwrite* keyword will continuously overwrite the output file with the latest output, so that it only contains one timestep worth of output. This option can only be used with the *ave running* setting.

The *title1* and *title2* and *title3* keywords allow specification of the strings that will be printed as the first 3 lines of the output file, assuming the *file* keyword was used. LIGGGHTS(R)-WSU uses default values for each of these, so they do not need to be specified.

By default, these header lines are as follows:

```
# Time-correlated data for fix ID
# TimeStep Number-of-time-windows
# Index TimeDelta Ncount valueI*valueJ valueI*valueJ ...
```

In the first line, ID is replaced with the fix-ID. The second line describes the two values that are printed at the first of each section of output. In the third line the value pairs are replaced with the appropriate fields from the fix ave/correlate command.

Let Sij = a set of time correlation data for input values I and J, namely the *Nrepeat* values:

```
Sij = Cij(0), Cij(Nevery), Cij(2*Nevery), ..., Cij(*Nrepeat-1)*Nevery)
```

As explained below, these datums are output as one column of a global array, which is effectively the correlation matrix.

The *trap* function defined for [equal-style variables](#) can be used to perform a time integration of this vector of datums, using a trapezoidal rule. This is useful for calculating various quantities which can be derived from time correlation data. If a normalization factor is needed for the time integration, it can be included in the variable formula or via the *prefactor* keyword.

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

No information about this fix is written to [binary restart files](#). None of the [fix_modify](#) options are relevant to this fix.

This fix computes a global array of values which can be accessed by various [output commands](#). The values can only be accessed on timesteps that are multiples of *Nfreq* since that is when averaging is performed. The global array has # of rows = *Nrepeat* and # of columns = Npair+2. The first column has the time delta (in timesteps) between the pairs of input values used to calculate the correlation, as described above. The 2nd column has the number of samples contributing to the correlation average, as described above. The remaining Npair columns are for I,J pairs of the N input values, as determined by the *type* keyword, as described above.

- For *type* = *auto*, the Npair = N columns are ordered: C11, C22, ..., CNN.
- For *type* = *upper*, the Npair = N*(N-1)/2 columns are ordered: C12, C13, ..., C1N, C23, ..., C2N, C34, ..., CN-1N.
- For *type* = *lower*, the Npair = N*(N-1)/2 columns are ordered: C21, C31, C32, C41, C42, C43, ..., CN1, CN2, ..., CNN-1.
- For *type* = *auto/upper*, the Npair = N*(N+1)/2 columns are ordered: C11, C12, C13, ..., C1N, C22, C23, ..., C2N, C33, C34, ..., CN-1N, CNN.
- For *type* = *auto/lower*, the Npair = N*(N+1)/2 columns are ordered: C11, C21, C22, C31, C32, C33, C41, ..., C44, CN1, CN2, ..., CNN-1, CNN.
- For *type* = *full*, the Npair = N^2 columns are ordered: C11, C12, ..., C1N, C21, C22, ..., C2N, C31, ..., C3N, ..., CN1, ..., CNN-1, CNN.

The array values calculated by this fix are treated as "intensive". If you need to divide them by the number of atoms, you must do this in a later processing step, e.g. when using them in a [variable](#).

No parameter of this fix can be used with the *start/stop* keywords of the [run](#) command. This fix is not invoked during [energy minimization](#).

Restrictions: none

Related commands:

compute, fix ave/time, fix ave/atom, fix ave/spatial, fix ave/histo, variable

Default: none

The option defaults are ave = one, type = auto, start = 0, no file output, title 1,2,3 = strings as described above, and prefactor = 1.0.

fix ave/euler command

Syntax:

```
fix ID group-ID ave/euler nevery N cell_size_relative c parallel par keywords values
```

- ID, group-ID are documented in fix command
- ave/euler = style name of this fix command
- nevery = obligatory keyword
- n = calculate average values every this many timesteps
- cell_size_relative = obligatory keyword
- c = cell size in multiples of max cutoff
- parallel = obligatory keyword
- par = "yes" or "no
- zero or more keyword/value pairs may be appended
- keyword = *basevolume_region*

```
    basevolume_region values = reg-ID
        region-ID = correct grid cell volume based on this region
```

Examples:

```
fix 1 all ave/euler nevery 100 cell_size_relative 4.5
```

Description:

Calculate cell-based averages of velocity, radius, volume fraction, and pressure (-1/3 * trace of the stress tensor) every few timesteps, as specified by the *nevery* keyword. The size of the cells is calculated as multiple of the maximum cutoff, via the *cell_size_relative*. Note that at least a relative cell size of 3 is required.

Note that velocity is favre (mass) averaged, whereas radius is arithmetically averaged. To calculate the stress, this command internally uses a compute stress/atom . It includes the convective term correctly for granular particles with non-zero average velocity (which is not included in compute stress/atom). However, it does not include bond, angle, diahedral or kspace contributions so that the stress tensor finally reads

$$S_{ab} = -\left[m(v_a - vave_a)(v_b - vave_b) + \frac{1}{2}\sum_{n=1}^{N_p}(r_{1_a}F_{1_b} + r_{2_a}F_{2_b}) \right]$$

where vave is the (cell-based) average velocity. The first term is a kinetic energy contribution for atom *I*. The second term is a pairwise energy contribution where *n* loops over the *Np* neighbors of atom *I*, *r1* and *r2* are the positions of the 2 atoms in the pairwise interaction, and *F1* and *F2* are the forces on the 2 atoms resulting from the pairwise interaction.

The *parallel* option determines if every process allocates its own local grid for postprocessing (for *parallel* = yes), or each proc contributes to one single global grid (for *parallel* = no). This will be slower since it requires parallel communication, but will ensure that the grid cells do not move over time (e.g. in case of a moving boundary)

The *basevolume_region* option allows to specify a region that represents the volume which can theoretically be filled with particles. This will then be used to correct the basis of the averaging volume for each cell in the

grid. For example, if you use a cylindrical wall, it makes sense to use an identical cylindrical region for the *basevolume_region* option, and the command will correctly calculate the volume fraction in the near-wall cells. the calculation of overlap between grid cells and the region is done using a Monte-Carlo approach.

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

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix.

This fix computes the above-mentioned quantities for output via a dump euler/vtk command. The values can only be accessed on timesteps that are multiples of *nevery* since that is when calculations are performed.

No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions:

Volume fractions and stresses are calculated based on the assumption of a structured (equidistant regular) grid, so volume fractions and stresses near walls that are not alligned with the grid will be incorrect.

Related commands:

compute, compute stress/atom, fix ave/atom, fix ave/histo, fix ave/time, fix ave/spatial,

Default: none

fix aveforce command

Syntax:

```
fix ID group-ID aveforce fx fy fz keyword value ...
```

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

> any of fx,fy,fz can be a variable (see below)
- zero or more keyword/value pairs may be appended to args
- keyword = *region*

```
  region value = region-ID
      region-ID = ID of region atoms must be in to have added force
```

Examples:

```
fix pressdown topwall aveforce 0.0 -1.0 0.0
fix 2 bottomwall aveforce NULL -1.0 0.0 region top
fix 2 bottomwall aveforce NULL -1.0 v_oscillate region top
```

Description:

Apply an additional external force to a group of atoms in such a way that every atom experiences the same force. This is useful for pushing on wall or boundary atoms so that the structure of the wall does not change over time.

The existing force is averaged for the group of atoms, component by component. The actual force on each atom is then set to the average value plus the component specified in this command. This means each atom in the group receives the same force.

Any of the fx,fy,fz values can be specified as NULL which means the force in that dimension is not changed. Note that this is not the same as specifying a 0.0 value, since that sets all forces to the same average value without adding in any additional force.

Any of the 3 quantities defining the force components can be specified as an equal-style [variable](#), namely *fx*, *fy*, *fz*. If the value is a variable, it should be specified as v_name, where name is the variable name. In this case, the variable will be evaluated each timestep, and its value used to determine the average force.

Equal-style variables can specify formulas with various mathematical functions, and include [thermo_style](#) command keywords for the simulation box parameters and timestep and elapsed time. Thus it is easy to specify a time-dependent average force.

If the *region* keyword is used, the atom must also be in the specified geometric [region](#) in order to have force added to it.

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

No information about this fix is written to [binary restart files](#). None of the [fix_modify](#) options are relevant to this fix.

This fix computes a global 3-vector of forces, which can be accessed by various output commands. This is the total force on the group of atoms before the forces on individual atoms are changed by the fix. The vector values calculated by this fix are "extensive".

No parameter of this fix can be used with the *start/stop* keywords of the run command.

The forces due to this fix are imposed during an energy minimization, invoked by the minimize command. You should not specify force components with a variable that has time-dependence for use with a minimizer, since the minimizer increments the timestep as the iteration count during the minimization.

Restrictions: none

Related commands:

fix setforce, fix addforce

Default: none

fix ave/histo command

Syntax:

```
fix ID group-ID ave/histo Nevery Nrepeat Nfreq lo hi Nbin value1 value2 ... keyword args ...
```

- ID, group-ID are documented in <u>fix</u> command
- ave/histo = style name of this fix command
- Nevery = use input values every this many timesteps
- Nrepeat = # of times to use input values for calculating histogram
- Nfreq = calculate histogram every this many timesteps
- lo,hi = lo/hi bounds within which to histogram
- Nbin = # of histogram bins
- one or more input values can be listed
- value = x, y, z, vx, vy, vz, fx, fy, fz, c_ID, c_ID[N], f_ID, f_ID[N], v_name

```
    x,y,z,vx,vy,vz,fx,fy,fz = atom attribute (position, velocity, force component)
    c_ID = scalar or vector calculated by a compute with ID
    c_ID[I] = Ith component of vector or Ith column of array calculated by a compute with ID
    f_ID = scalar or vector calculated by a fix with ID
    f_ID[I] = Ith component of vector or Ith column of array calculated by a fix with ID
    v_name = value(s) calculated by an equal-style or atom-style variable with name
```

- zero or more keyword/arg pairs may be appended
- keyword = *mode* or *file* or *ave* or *start* or *beyond* or *overwrite* or *title1* or *title2* or *title3*

```
  mode arg = scalar or vector
     scalar = all input values are scalars
     vector = all input values are vectors
  file arg = filename
     filename = name of file to output histogram(s) to
  ave args = one or running or window
     one = output a new average value every Nfreq steps
     running = output cumulative average of all previous Nfreq steps
     window M = output average of M most recent Nfreq steps
  start args = Nstart
     Nstart = start averaging on this timestep
  beyond arg = ignore or end or extra
     ignore = ignore values outside histogram lo/hi bounds
     end = count values outside histogram lo/hi bounds in end bins
     extra = create 2 extra bins for value outside histogram lo/hi bounds
  overwrite arg = none = overwrite output file with only latest output
  title1 arg = string
     string = text to print as 1st line of output file
  title2 arg = string
     string = text to print as 2nd line of output file
  title3 arg = string
     string = text to print as 3rd line of output file, only for vector mode
```

Examples:

```
fix 1 all ave/histo 100 5 1000 0.5 1.5 50 c_myTemp file temp.histo ave running
fix 1 all ave/histo 100 5 1000 -5 5 100 c_thermo_press[2] c_thermo_press[3] title1 "My output val
fix 1 all ave/histo 1 100 1000 -2.0 2.0 18 vx vy vz mode vector ave running beyond extra
```

Description:

fix ave/histo command   162

Use one or more values as inputs every few timesteps, histogram them, and average the histogram over longer timescales. The resulting histogram can be used by other output commands, and can also be written to a file.

The group specified with this command is ignored for global and local input values. For per-atom input values, only atoms in the group contribute to the histogram. Note that regardless of the specified group, specified values may represent calculations performed by computes and fixes which store their own "group" definition.

A histogram is simply a count of the number of values that fall within a histogram bin. *Nbins* are defined, with even spacing between *lo* and *hi*. Values that fall outside the lo/hi bounds can be treated in different ways; see the discussion of the *beyond* keyword below.

Each input value can be an atom attribute (position, velocity, force component) or can be the result of a compute or fix or the evaluation of an equal-style or atom-style variable. The set of input values can be either all global, all per-atom, or all local quantities. Inputs of different kinds (e.g. global and per-atom) cannot be mixed. Atom attributes are per-atom vector values. See the doc page for individual "compute" and "fix" commands to see what kinds of quantities they generate.

The input values must either be all scalars or all vectors (or arrays), depending on the setting of the *mode* keyword.

If *mode* = vector, then the input values may either be vectors or arrays. If a global array is listed, then it is the same as if the individual columns of the array had been listed one by one. E.g. these 2 fix ave/histo commands are equivalent, since the compute com/molecule command creates a global array with 3 columns:

```
compute myCOM all com/molecule
fix 1 all ave/histo 100 1 100 c_myCOM file tmp1.com mode vector
fix 2 all ave/histo 100 1 100 c_myCOM[1] c_myCOM[2] c_myCOM[3] file tmp2.com mode vector
```

The output of this command is a single histogram for all input values combined together, not one histogram per input value. See below for details on the format of the output of this fix.

The *Nevery*, *Nrepeat*, and *Nfreq* arguments specify on what timesteps the input values will be used in order to contribute to the histogram. The final histogram is generated on timesteps that are multiple of *Nfreq*. It is averaged over *Nrepeat* histograms, computed in the preceding portion of the simulation every *Nevery* timesteps. *Nfreq* must be a multiple of *Nevery* and *Nevery* must be non-zero even if *Nrepeat* is 1. Also, the timesteps contributing to the histogram cannot overlap, i.e. Nfreq > (Nrepeat-1)*Nevery is required.

For example, if Nevery=2, Nrepeat=6, and Nfreq=100, then input values on timesteps 90,92,94,96,98,100 will be used to compute the final histogram on timestep 100. Similarly for timesteps 190,192,194,196,198,200 on timestep 200, etc. If Nrepeat=1 and Nfreq = 100, then no time averaging of the histogram is done; a histogram is simply generated on timesteps 100,200,etc.

The atom attribute values (x,y,z,vx,vy,vz,fx,fy,fz) are self-explanatory. Note that other atom attributes can be used as inputs to this fix by using the compute property/atom command and then specifying an input value from that compute.

If a value begins with "c_", a compute ID must follow which has been previously defined in the input script. If *mode* = scalar, then if no bracketed term is appended, the global scalar calculated by the compute is used. If a bracketed term is appended, the Ith element of the global vector calculated by the compute is used. If *mode* = vector, then if no bracketed term is appended, the global or per-atom or local vector calculated by the compute is used. Or if the compute calculates an array, all of the columns of the array are used as if they had been specified as individual vectors (see description above). If a bracketed term is appended, the Ith column of the global or per-atom or local array calculated by the compute is used.

Note that there is a [compute reduce](#) command which can sum per-atom quantities into a global scalar or vector which can thus be accessed by fix ave/histo. Or it can be a compute defined not in your input script, but by [thermodynamic output](#) or other fixes such as [fix nvt](#) or [fix temp/rescale](#). See the doc pages for these commands which give the IDs of these computes. Users can also write code for their own compute styles and [add them to LIGGGHTS(R)-WSU](#).

If a value begins with "f_", a fix ID must follow which has been previously defined in the input script. If *mode* = scalar, then if no bracketed term is appended, the global scalar calculated by the fix is used. If a bracketed term is appended, the Ith element of the global vector calculated by the fix is used. If *mode* = vector, then if no bracketed term is appended, the global or per-atom or local vector calculated by the fix is used. Or if the fix calculates an array, all of the columns of the array are used as if they had been specified as individual vectors (see description above). If a bracketed term is appended, the Ith column of the global or per-atom or local array calculated by the fix is used.

Note that some fixes only produce their values on certain timesteps, which must be compatible with *Nevery*, else an error will result. Users can also write code for their own fix styles and [add them to LIGGGHTS(R)-WSU](#).

If a value begins with "v_", a variable name must follow which has been previously defined in the input script. If *mode* = scalar, then only equal-style variables can be used, which produce a global value. If *mode* = vector, then only atom-style variables can be used, which produce a per-atom vector. See the [variable](#) command for details. Note that variables of style *equal* and *atom* define a formula which can reference individual atom properties or thermodynamic keywords, or they can invoke other computes, fixes, or variables when they are evaluated, so this is a very general means of specifying quantities to histogram.

Additional optional keywords also affect the operation of this fix.

If the *mode* keyword is set to *scalar*, then all input values must be global scalars, or elements of global vectors. If the *mode* keyword is set to *vector*, then all input values must be global or per-atom or local vectors, or columns of global or per-atom or local arrays.

The *beyond* keyword determines how input values that fall outside the *lo* to *hi* bounds are treated. Values such that *lo* <= value <= *hi* are assigned to one bin. Values on a bin boundary are assigned to the lower of the 2 bins. If *beyond* is set to *ignore* then values < *lo* and values > *hi* are ignored, i.e. they are not binned. If *beyond* is set to *end* then values < *lo* are counted in the first bin and values > *hi* are counted in the last bin. If *beyond* is set to *extend* then two extra bins are created, so that there are Nbins+2 total bins. Values < *lo* are counted in the first bin and values > *hi* are counted in the last bin (Nbins+1). Values between *lo* and *hi* (inclusive) are counted in bins 2 thru Nbins+1. The "coordinate" stored and printed for these two extra bins is *lo* and *hi*.

The *ave* keyword determines how the histogram produced every *Nfreq* steps are averaged with histograms produced on previous steps that were multiples of *Nfreq*, before they are accessed by another output command or written to a file.

If the *ave* setting is *one*, then the histograms produced on timesteps that are multiples of *Nfreq* are independent of each other; they are output as-is without further averaging.

If the *ave* setting is *running*, then the histograms produced on timesteps that are multiples of *Nfreq* are summed and averaged in a cumulative sense before being output. Each bin value in the histogram is thus the average of the bin value produced on that timestep with all preceding values for the same bin. This running average begins when the fix is defined; it can only be restarted by deleting the fix via the [unfix](#) command, or by re-defining the fix by re-specifying it.

If the *ave* setting is *window*, then the histograms produced on timesteps that are multiples of *Nfreq* are summed within a moving "window" of time, so that the last M histograms are used to produce the output. E.g.

if M = 3 and Nfreq = 1000, then the output on step 10000 will be the combined histogram of the individual histograms on steps 8000,9000,10000. Outputs on early steps will be sums over less than M histograms if they are not available.

The *start* keyword specifies what timestep histogramming will begin on. The default is step 0. Often input values can be 0.0 at time 0, so setting *start* to a larger value can avoid including a 0.0 in a running or windowed histogram.

The *file* keyword allows a filename to be specified. Every *Nfreq* steps, one histogram is written to the file. This includes a leading line that contains the timestep, number of bins, the total count of values contributing to the histogram, the count of values that were not histogrammed (see the *beyond* keyword), the minimum value encountered, and the maximum value encountered. The min/max values include values that were not histogrammed. Following the leading line, one line per bin is written into the file. Each line contains the bin #, the coordinate for the center of the bin (between *lo* and *hi*), the count of values in the bin, and the normalized count. The normalized count is the bin count divided by the total count (not including values not histogrammed), so that the normalized values sum to 1.0 across all bins.

The *overwrite* keyword will continuously overwrite the output file with the latest output, so that it only contains one timestep worth of output. This option can only be used with the *ave running* setting.

The *title1* and *title2* and *title3* keywords allow specification of the strings that will be printed as the first 3 lines of the output file, assuming the *file* keyword was used. LIGGGHTS(R)-WSU uses default values for each of these, so they do not need to be specified.

By default, these header lines are as follows:

```
# Histogram for fix ID
# TimeStep Number-of-bins Total-counts Missing-counts Min-value Max-value
# Bin Coord Count Count/Total
```

In the first line, ID is replaced with the fix-ID. The second line describes the six values that are printed at the first of each section of output. The third describes the 4 values printed for each bin in the histogram.

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

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix.

This fix produces a global vector and global array which can be accessed by various output commands. The values can only be accessed on timesteps that are multiples of *Nfreq* since that is when a histogram is generated. The global vector has 4 values:

- 1 = total counts in the histogram
- 2 = values that were not histogrammed (see *beyond* keyword)
- 3 = min value of all input values, including ones not histogrammed
- 4 = max value of all input values, including ones not histogrammed

The global array has # of rows = Nbins and # of columns = 3. The first column has the bin coordinate, the 2nd column has the count of values in that histogram bin, and the 3rd column has the bin count divided by the total count (not including missing counts), so that the values in the 3rd column sum to 1.0.

The vector and array values calculated by this fix are all treated as "intensive". If this is not the case, e.g. due to histogramming per-atom input values, then you will need to account for that when interpreting the values produced by this fix.

No parameter of this fix can be used with the *start/stop* keywords of the [run](run) command. This fix is not invoked during [energy minimization](energy_minimization).

Restrictions: none

Related commands:

compute, [fix ave/atom](fix_ave_atom), [fix ave/spatial](fix_ave_spatial), [fix ave/time](fix_ave_time), variable, [fix ave/correlate](fix_ave_correlate),

Default: none

The option defaults are mode = scalar, ave = one, start = 0, no file output, beyond = ignore, and title 1,2,3 = strings as described above.

fix ave/spatial command

Syntax:

```
fix ID group-ID ave/spatial Nevery Nrepeat Nfreq dim origin delta ... value1 value2 ... keyword a
```

- ID, group-ID are documented in [fix](#) command
- ave/spatial = style name of this fix command
- Nevery = use input values every this many timesteps
- Nrepeat = # of times to use input values for calculating averages
- Nfreq = calculate averages every this many timesteps
- dim, origin, delta can be repeated 1, 2, or 3 times for 1d, 2d, or 3d bins

```
dim = x or y or z
origin = lower or center or upper or coordinate value (distance units)
delta = thickness of spatial bins in dim (distance units)
```

- one or more input values can be listed
- value = vx, vy, vz, fx, fy, fz, density/mass, density/number, c_ID, c_ID[I], f_ID, f_ID[I], v_name

```
vx,vy,vz,fx,fy,fz = atom attribute (velocity, force component)
density/number, density/mass = number or mass density
c_ID = per-atom vector calculated by a compute with ID
c_ID[I] = Ith column of per-atom array calculated by a compute with ID
f_ID = per-atom vector calculated by a fix with ID
f_ID[I] = Ith column of per-atom array calculated by a fix with ID
v_name = per-atom vector calculated by an atom-style variable with name
```

- zero or more keyword/arg pairs may be appended
- keyword = *norm* or *units* or *file* or *ave* or *overwrite* or *title1* or *title2* or *title3* or *write_ts* or *std*

```
  units arg = box or lattice or reduced
    norm arg = all or sample
  region arg = region-ID
    region-ID = ID of region atoms must be in to contribute to spatial averaging
  ave args = one or running or window M
    one = output new average value every Nfreq steps
    running = output cumulative average of all previous Nfreq steps
    window M = output average of M most recent Nfreq steps
  file arg = filename
    filename = file to write results to
  overwrite arg = none = overwrite output file with only latest output
  title1 arg = string
    string = text to print as 1st line of output file
  title2 arg = string
    string = text to print as 2nd line of output file
  title3 arg = string
    string = text to print as 3rd line of output file
  write_ts arg = yes or no
    yes or no = do nor do not write time-step info and number of samples to file
  std arg = N1 N2 filename
        N1 = lower limit of particle number per bin for sampling
        N2 = upper limit of particle number per bin for sampling
        filename = file to write results into
```

Examples:

```
fix 1 all ave/spatial 10000 1 10000 z lower 0.02 c_myCentro units reduced &
                   title1 "My output values"
fix 1 flow ave/spatial 100 10 1000 y 0.0 1.0 vx vz norm sample file vel.profile
```

```
fix 1 flow ave/spatial 100 5 1000 z lower 1.0 y 0.0 2.5 density/mass ave running
fix 1 all ave/spatial 1000 1 1000 x 0 1e-3 y 0 1e-3 z 0 1e-3 f_tracer[0] file bin_data.dat std 12
```

Description:

Use one or more per-atom vectors as inputs every few timesteps, bin their values spatially into 1d, 2d, or 3d bins based on current atom coordinates, and average the bin values over longer timescales. The resulting bin averages can be used by other [output commands](#) such as [thermo_style custom](#), and can also be written to a file.

The group specified with the command means only atoms within the group contribute to bin averages. If the *region* keyword is used, the atom must be in both the group and the specified geometric [region](#) in order to contribute to bin averages.

Each listed value can be an atom attribute (position, velocity, force component), a mass or number density, or the result of a [compute](#) or [fix](#) or the evaluation of an atom-style [variable](#). In the latter cases, the compute, fix, or variable must produce a per-atom quantity, not a global quantity. If you wish to time-average global quantities from a compute, fix, or variable, then see the [fix ave/time](#) command.

[Computes](#) that produce per-atom quantities are those which have the word *atom* in their style name. See the doc pages for individual [fixes](#) to determine which ones produce per-atom quantities. [Variables](#) of style *atom* are the only ones that can be used with this fix since all other styles of variable produce global quantities.

The per-atom values of each input vector are binned and averaged independently of the per-atom values in other input vectors.

The size and dimensionality of the bins (1d = layers or slabs, 2d = pencils, 3d = boxes) are determined by the *dim*, *origin*, and *delta* settings and how many times they are specified (1, 2, or 3). See details below.

IMPORTANT NOTE: This fix works by creating an array of size Nbins by Nvalues on each processor. Nbins is the total number of bins; Nvalues is the number of input values specified. Each processor loops over its atoms, tallying its values to the appropriate bin. Then the entire array is summed across all processors. This means that using a large number of bins (easy to do for 2d or 3d bins) will incur an overhead in memory and computational cost (summing across processors), so be careful to use reasonable numbers of bins.

The *Nevery*, *Nrepeat*, and *Nfreq* arguments specify on what timesteps the input values will be used to bin them and contribute to the average. The final averaged quantities are generated on timesteps that are a multiples of *Nfreq*. The average is over *Nrepeat* quantities, computed in the preceding portion of the simulation every *Nevery* timesteps. *Nfreq* must be a multiple of *Nevery* and *Nevery* must be non-zero even if *Nrepeat* is 1. Also, the timesteps contributing to the average value cannot overlap, i.e. Nfreq > (Nrepeat-1)*Nevery is required.

For example, if Nevery=2, Nrepeat=6, and Nfreq=100, then values on timesteps 90,92,94,96,98,100 will be used to compute the final average on timestep 100. Similarly for timesteps 190,192,194,196,198,200 on timestep 200, etc. If Nrepeat=1 and Nfreq = 100, then no time averaging is done; values are simply generated on timesteps 100,200,etc.

Each per-atom property is also averaged over atoms in each bin. Bins can be 1d layers or slabs, 2d pencils, or 3d boxes. This depends on how many times (1, 2, or 3) the *dim*, *origin*, and *delta* settings are specified in the fix ave/spatial command. For 2d or 3d bins, there is no restriction on specifying dim = x before dim = y, or dim = y before dim = z. Bins in a particular *dim* have a bin size in that dimension given by *delta*. Every Nfreq steps, when averaging is being performed and the per-atom property is calculated for the first time, the number of bins and the bin sizes and boundaries are computed. Thus if the simulation box changes size during a simulation, the number of bins and their boundaries may also change. In each dimension, bins are defined relative to a specified *origin*, which may be the lower/upper edge of the simulation box (in *dim*) or its center

fix ave/spatial command

point, or a specified coordinate value. Starting at the origin, sufficient bins are created in both directions to completely cover the box. On subsequent timesteps every atom is mapped to one of the bins. Atoms beyond the lowermost/uppermost bin in a dimension are counted in the first/last bin in that dimension.

For orthogonal simulation boxes, the bins are also layers, pencils, or boxes aligned with the xyz coordinate axes. For triclinic (non-orthogonal) simulation boxes, the bins are so that they are parallel to the tilted faces of the simulation box. See this section of the manual for a discussion of the geometry of triclinic boxes in LIGGGHTS(R)-WSU. As described there, a tilted simulation box has edge vectors a,b,c. In that nomenclature, bins in the x dimension have faces with normals in the "b" cross "c" direction. Bins in y have faces normal to the "a" cross "c" direction. And bins in z have faces normal to the "a" cross "b" direction. Note that in order to define the size and position of these bins in an unambiguous fashion, the *units* option must be set to *reduced* when using a triclinic simulation box, as noted below.

The atom attribute values (vx,vy,vz,fx,fy,fz) are self-explanatory. Note that other atom attributes (including atom postitions x,y,z) can be used as inputs to this fix by using the compute property/atom command and then specifying an input value from that compute.

The *density/number* value means the number density is computed in each bin, i.e. a weighting of 1 for each atom. The *density/mass* value means the mass density is computed in each bind, i.e. each atom is weighted by its mass. The resulting density is normalized by the volume of the bin so that units of number/volume or density are output. See the units command doc page for the definition of density for each choice of units, e.g. gram/cm^3.

If a value begins with "c_", a compute ID must follow which has been previously defined in the input script. If no bracketed integer is appended, the per-atom vector calculated by the compute is used. If a bracketed integer is appended, the Ith column of the per-atom array calculated by the compute is used. Users can also write code for their own compute styles and add them to LIGGGHTS(R)-WSU.

If a value begins with "f_", a fix ID must follow which has been previously defined in the input script. If no bracketed integer is appended, the per-atom vector calculated by the fix is used. If a bracketed integer is appended, the Ith column of the per-atom array calculated by the fix is used. Note that some fixes only produce their values on certain timesteps, which must be compatible with *Nevery*, else an error results. Users can also write code for their own fix styles and add them to LIGGGHTS(R)-WSU.

If a value begins with "v_", a variable name must follow which has been previously defined in the input script. Variables of style *atom* can reference thermodynamic keywords and various per-atom attributes, or invoke other computes, fixes, or variables when they are evaluated, so this is a very general means of generating per-atom quantities to spatially average.

Additional optional keywords also affect the operation of this fix.

The *units* keyword determines the meaning of the distance units used for the bin size *delta* and for *origin* if it is a coordinate value. For orthogonal simulation boxes, any of the 3 options may be used. For non-orthogonal (triclinic) simulation boxes, only the *reduced* option may be used.

A *box* value selects standard distance units as defined by the units command, e.g. Angstroms for units = real or metal. A *lattice* value means the distance units are in lattice spacings. The lattice command must have been previously used to define the lattice spacing. A *reduced* value means normalized unitless values between 0 and 1, which represent the lower and upper faces of the simulation box respectively. Thus an *origin* value of 0.5 means the center of the box in any dimension. A *delta* value of 0.1 means 10 bins span the box in that dimension.

Consider a non-orthogonal box, with bins that are 1d layers or slabs in the x dimension. No matter how the box is tilted, an *origin* of 0.0 means start layers at the lower "b" cross "c" plane of the simulation box and an

origin of 1.0 means to start layers at the upper "b" cross "c" face of the box. A *delta* value of 0.1 means there will be 10 layers from 0.0 to 1.0, regardless of the current size or shape of the simulation box.

The *norm* keyword affects how averaging is done for the output produced every *Nfreq* timesteps. For an *all* setting, a bin quantity is summed over all atoms in all *Nrepeat* samples, as is the count of atoms in the bin. The printed value for the bin is Total-quantity / Total-count. In other words it is an average over the entire *Nfreq* timescale.

For a *sample* setting, the bin quantity is summed over atoms for only a single sample, as is the count, and a "average sample value" is computed, i.e. Sample-quantity / Sample-count. The printed value for the bin is the average of the *Nrepeat* "average sample values", In other words it is an average of an average.

The *ave* keyword determines how the bin values produced every *Nfreq* steps are averaged with bin values produced on previous steps that were multiples of *Nfreq*, before they are accessed by another output command or written to a file.

If the *ave* setting is *one*, then the bin values produced on timesteps that are multiples of *Nfreq* are independent of each other; they are output as-is without further averaging.

If the *ave* setting is *running*, then the bin values produced on timesteps that are multiples of *Nfreq* are summed and averaged in a cumulative sense before being output. Each output bin value is thus the average of the bin value produced on that timestep with all preceding values for the same bin. This running average begins when the fix is defined; it can only be restarted by deleting the fix via the unfix command, or re-defining the fix by re-specifying it.

If the *ave* setting is *window*, then the bin values produced on timesteps that are multiples of *Nfreq* are summed and averaged within a moving "window" of time, so that the last M values for the same bin are used to produce the output. E.g. if M = 3 and Nfreq = 1000, then the output on step 10000 will be the average of the individual bin values on steps 8000,9000,10000. Outputs on early steps will average over less than M values if they are not available.

The *file* keyword allows a filename to be specified. Every *Nfreq* timesteps, a section of bin info will be written to a text file in the following format. A line with the timestep and number of bin is written. Output of this line can be supressed with the *write_ts* keyword. Then one line per bin is written, containing the bin ID (1-N), the coordinate of the center of the bin, the number of atoms in the bin, and one or more calculated values. The number of values in each line corresponds to the number of values specified in the fix ave/spatial command. The number of atoms and the value(s) are average quantities. If the value of the *units* keyword is *box* or *lattice*, the "coord" is printed in box units. If the value of the *units* keyword is *reduced*, the "coord" is printed in reduced units (0-1).

The *overwrite* keyword will continuously overwrite the output file with the latest output, so that it only contains one timestep worth of output. This option can only be used with the *ave running* setting.

The *title1* and *title2* and *title3* keywords allow specification of the strings that will be printed as the first 3 lines of the output file, assuming the *file* keyword was used. LIGGGHTS(R)-WSU uses default values for each of these, so they do not need to be specified. If either of them is specified as "", then the line is omitted

By default, these header lines are as follows:

```
# Spatial-averaged data for fix ID and group name
# Timestep Number-of-bins
# Bin Coord1 Coord2 Coord3 Count value1 value2 ...
```

In the first line, ID and name are replaced with the fix-ID and group name. The second line describes the two values that are printed at the first of each section of output. In the third line the values are replaced with the

appropriate fields from the fix ave/spatial command. The Coord2 and Coord3 entries in the third line only appear for 2d and 3d bins respectively. For 1d bins, the word Coord1 is replaced by just Coord.

If the *std* keyword is set, mean and standard deviation of the specified values (value1, value2, etc.) over samples of a defined size are calculated. The sample size has to be defined by a lower limit (N1) and an upper limit (N2>N1). All bins containing a particle count between N1 and N2 (including N1 and N2) are used as samples. Every *Nfreq* timestep a line is written to a file specified after N1 and N2, including the following numbers: timestep, total number of atoms, total number of bins, maximum number of atoms per bin, number of empty bins, number of bins including less atoms than N1, number of bins including more atoms than N2, number of samples, average number of atoms per sample, followed by three quantities for each defined value: true average (over all atoms), average over the chosen samples, standard deviation over the chosen samples. For the calculation of the standard deviation the (known) true average is used instead of the samples average (the latter is only an estimate for the true average!).

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

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix.

This fix computes a global array of values which can be accessed by various output commands. The values can only be accessed on timesteps that are multiples of *Nfreq* since that is when averaging is performed. The global array has # of rows = Nbins and # of columns = Ndim+1+Nvalues, where Ndim = 1,2,3 for 1d,2d,3d bins. The first 1 or 2 or 3 columns have the bin coordinates (center of the bin) in the appropriate dimensions, the next column has the count of atoms in that bin, and the remaining columns are the Nvalue quantities. When the array is accessed with an I that exceeds the current number of bins, than a 0.0 is returned by the fix instead of an error, since the number of bins can vary as a simulation runs, depending on the simulation box size. 2d or 3d bins are ordered so that the last dimension(s) vary fastest. The array values calculated by this fix are "intensive", since they are already normalized by the count of atoms in each bin.

No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions:

When the *ave* keyword is set to *running* or *window* then the number of bins must remain the same during the simulation, so that the appropriate averaging can be done. This will be the case if the simulation box size doesn't change or if the *units* keyword is set to *reduced*.

Related commands:

compute, fix ave/atom, fix ave/histo, fix ave/time, variable, fix ave/correlate,

Default:

The option defaults are units = box, norm = all, no file output, and ave = one, title 1,2,3 = strings as described above.

fix ave/time command

Syntax:

```
fix ID group-ID ave/time Nevery Nrepeat Nfreq value1 value2 ... keyword args ...
```

- ID, group-ID are documented in fix command
- ave/time = style name of this fix command
- Nevery = use input values every this many timesteps
- Nrepeat = # of times to use input values for calculating averages
- Nfreq = calculate averages every this many timesteps
- one or more input values can be listed
- value = c_ID, c_ID[N], f_ID, f_ID[N], v_name

```
    c_ID = global scalar or vector calculated by a compute with ID
    c_ID[I] = Ith component of global vector or Ith column of global array calculated by a c
    f_ID = global scalar or vector calculated by a fix with ID
    f_ID[I] = Ith component of global vector or Ith column of global array calculated by a f
    v_name = global value calculated by an equal-style variable with name
```

- zero or more keyword/arg pairs may be appended
- keyword = *mode* or *file* or *ave* or *start* or *off* or *overwrite* or *title1* or *title2* or *title3*

```
  mode arg = scalar or vector
      scalar = all input values are global scalars
      vector = all input values are global vectors or global arrays
    ave args = one or running or window M
      one = output a new average value every Nfreq steps
      running = output cummulative average of all previous Nfreq steps
      window M = output average of M most recent Nfreq steps
    start args = Nstart
      Nstart = start averaging on this timestep
    off arg = M = do not average this value
      M = value # from 1 to Nvalues
    file arg = filename
      filename = name of file to output time averages to
    overwrite arg = none = overwrite output file with only latest output
    title1 arg = string
      string = text to print as 1st line of output file
    title2 arg = string
      string = text to print as 2nd line of output file
    title3 arg = string
      string = text to print as 3rd line of output file, only for vector mode
```

Examples:

```
fix 1 all ave/time 100 5 1000 c_myTemp c_thermo_temp file temp.profile
fix 1 all ave/time 100 5 1000 c_thermo_press[2] ave window 20 &
                            title1 "My output values"
fix 1 all ave/time 1 100 1000 f_indent f_indent[1] file temp.indent off 1
```

Description:

Use one or more global values as inputs every few timesteps, and average them over longer timescales. The resulting averages can be used by other output commands such as thermo_style custom, and can also be written to a file. Note that if no time averaging is done, this command can be used as a convenient way to simply output one or more global values to a file.

The group specified with this command is ignored. However, note that specified values may represent calculations performed by computes and fixes which store their own "group" definitions.

Each listed value can be the result of a compute or fix or the evaluation of an equal-style variable. In each case, the compute, fix, or variable must produce a global quantity, not a per-atom or local quantity. If you wish to spatial- or time-average or histogram per-atom quantities from a compute, fix, or variable, then see the fix ave/spatial, fix ave/atom, or fix ave/histo commands. If you wish to sum a per-atom quantity into a single global quantity, see the compute reduce command.

Computes that produce global quantities are those which do not have the word *atom* in their style name. Only a few fixes produce global quantities. See the doc pages for individual fixes for info on which ones produce such values. Variables of style *equal* are the only ones that can be used with this fix. Variables of style *atom* cannot be used, since they produce per-atom values.

The input values must either be all scalars or all vectors (or arrays), depending on the setting of the *mode* keyword. In both cases, the averaging is performed independently on each input value. I.e. each input scalar is averaged independently and each element of each input vector (or array) is averaged independently.

If *mode* = vector, then the input values may either be vectors or arrays and all must be the same "length", which is the length of the vector or number of rows in the array. If a global array is listed, then it is the same as if the individual columns of the array had been listed one by one. E.g. these 2 fix ave/time commands are equivalent, since the compute rdf command creates, in this case, a global array with 3 columns, each of length 50:

```
compute myRDF all rdf 50 1 2
fix 1 all ave/time 100 1 100 c_myRDF file tmp1.rdf mode vector
fix 2 all ave/time 100 1 100 c_myRDF[1] c_myRDF[2] c_myRDF[3] file tmp2.rdf mode vector
```

The *Nevery*, *Nrepeat*, and *Nfreq* arguments specify on what timesteps the input values will be used in order to contribute to the average. The final averaged quantities are generated on timesteps that are a mlutiple of *Nfreq*. The average is over *Nrepeat* quantities, computed in the preceding portion of the simulation every *Nevery* timesteps. *Nfreq* must be a multiple of *Nevery* and *Nevery* must be non-zero even if *Nrepeat* is 1. Also, the timesteps contributing to the average value cannot overlap, i.e. Nfreq > (Nrepeat-1)*Nevery is required.

For example, if Nevery=2, Nrepeat=6, and Nfreq=100, then values on timesteps 90,92,94,96,98,100 will be used to compute the final average on timestep 100. Similarly for timesteps 190,192,194,196,198,200 on timestep 200, etc. If Nrepeat=1 and Nfreq = 100, then no time averaging is done; values are simply generated on timesteps 100,200,etc.

If a value begins with "c_", a compute ID must follow which has been previously defined in the input script. If *mode* = scalar, then if no bracketed term is appended, the global scalar calculated by the compute is used. If a bracketed term is appended, the Ith element of the global vector calculated by the compute is used. If *mode* = vector, then if no bracketed term is appended, the global vector calculated by the compute is used. Or if the compute calculates an array, all of the columns of the global array are used as if they had been specified as individual vectors (see description above). If a bracketed term is appended, the Ith column of the global array calculated by the compute is used.

Note that there is a compute reduce command which can sum per-atom quantities into a global scalar or vector which can thus be accessed by fix ave/time. Or it can be a compute defined not in your input script, but by thermodynamic output or other fixes such as fix nvt or fix temp/rescale. See the doc pages for these commands which give the IDs of these computes. Users can also write code for their own compute styles and add them to LIGGGHTS(R)-WSU.

If a value begins with "f_", a fix ID must follow which has been previously defined in the input script. If *mode* = scalar, then if no bracketed term is appended, the global scalar calculated by the fix is used. If a bracketed

term is appended, the Ith element of the global vector calculated by the fix is used. If *mode* = vector, then if no bracketed term is appended, the global vector calculated by the fix is used. Or if the fix calculates an array, all of the columns of the global array are used as if they had been specified as individual vectors (see description above). If a bracketed term is appended, the Ith column of the global array calculated by the fix is used.

Note that some fixes only produce their values on certain timesteps, which must be compatible with *Nevery*, else an error will result. Users can also write code for their own fix styles and add them to LIGGGHTS(R)-WSU.

If a value begins with "v_", a variable name must follow which has been previously defined in the input script. Variables can only be used as input for *mode* = scalar. Only equal-style variables can be referenced. See the variable command for details. Note that variables of style *equal* define a formula which can reference individual atom properties or thermodynamic keywords, or they can invoke other computes, fixes, or variables when they are evaluated, so this is a very general means of specifying quantities to time average.

Additional optional keywords also affect the operation of this fix.

If the *mode* keyword is set to *scalar*, then all input values must be global scalars, or elements of global vectors. If the *mode* keyword is set to *vector*, then all input values must be global vectors, or columns of global arrays. They can also be global arrays, which are converted into a series of global vectors (one per column), as explained above.

The *ave* keyword determines how the values produced every *Nfreq* steps are averaged with values produced on previous steps that were multiples of *Nfreq*, before they are accessed by another output command or written to a file.

If the *ave* setting is *one*, then the values produced on timesteps that are multiples of *Nfreq* are independent of each other; they are output as-is without further averaging.

If the *ave* setting is *running*, then the values produced on timesteps that are multiples of *Nfreq* are summed and averaged in a cummulative sense before being output. Each output value is thus the average of the value produced on that timestep with all preceding values. This running average begins when the fix is defined; it can only be restarted by deleting the fix via the unfix command, or by re-defining the fix by re-specifying it.

If the *ave* setting is *window*, then the values produced on timesteps that are multiples of *Nfreq* are summed and averaged within a moving "window" of time, so that the last M values are used to produce the output. E.g. if M = 3 and Nfreq = 1000, then the output on step 10000 will be the average of the individual values on steps 8000,9000,10000. Outputs on early steps will average over less than M values if they are not available.

The *start* keyword specifies what timestep averaging will begin on. The default is step 0. Often input values can be 0.0 at time 0, so setting *start* to a larger value can avoid including a 0.0 in a running or windowed average.

The *off* keyword can be used to flag any of the input values. If a value is flagged, it will not be time averaged. Instead the most recent input value will always be stored and output. This is useful if one of more of the inputs produced by a compute or fix or variable are effectively constant or are simply current values. E.g. they are being written to a file with other time-averaged values for purposes of creating well-formatted output.

The *file* keyword allows a filename to be specified. Every *Nfreq* steps, one quantity or vector of quantities is written to the file for each input value specified in the fix ave/time command. For *mode* = scalar, this means a single line is written each time output is performed. Thus the file ends up to be a series of lines, i.e. one column of numbers for each input value. For *mode* = vector, an array of numbers is written each time output is performed. The number of rows is the length of the input vectors, and the number of columns is the number of values. Thus the file ends up to be a series of these array sections.

The *overwrite* keyword will continuously overwrite the output file with the latest output, so that it only contains one timestep worth of output. This option can only be used with the *ave running* setting.

The *title1* and *title2* and *title3* keywords allow specification of the strings that will be printed as the first 2 or 3 lines of the output file, assuming the *file* keyword was used. LIGGGHTS(R)-WSU uses default values for each of these, so they do not need to be specified.

By default, these header lines are as follows for *mode* = scalar:

```
# Time-averaged data for fix ID
# TimeStep value1 value2 ...
```

In the first line, ID is replaced with the fix-ID. In the second line the values are replaced with the appropriate fields from the fix ave/time command. There is no third line in the header of the file, so the *title3* setting is ignored when *mode* = scalar.

By default, these header lines are as follows for *mode* = vector:

```
# Time-averaged data for fix ID
# TimeStep Number-of-rows
# Row value1 value2 ...
```

In the first line, ID is replaced with the fix-ID. The second line describes the two values that are printed at the first of each section of output. In the third line the values are replaced with the appropriate fields from the fix ave/time command.

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

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix.

This fix produces a global scalar or global vector or global array which can be accessed by various output commands. The values can only be accessed on timesteps that are multiples of *Nfreq* since that is when averaging is performed.

A scalar is produced if only a single input value is averaged and *mode* = scalar. A vector is produced if multiple input values are averaged for *mode* = scalar, or a single input value for *mode* = vector. In the first case, the length of the vector is the number of inputs. In the second case, the length of the vector is the same as the length of the input vector. An array is produced if multiple input values are averaged and *mode* = vector. The global array has # of rows = length of the input vectors and # of columns = number of inputs.

If the fix prouduces a scalar or vector, then the scalar and each element of the vector can be either "intensive" or "extensive". If the fix produces an array, then all elements in the array must be the same, either "intensive" or "extensive". If a compute or fix provides the value being time averaged, then the compute or fix determines whether the value is intensive or extensive; see the doc page for that compute or fix for further info. Values produced by a variable are treated as intensive.

No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions: none

Related commands:

compute, fix ave/atom, fix ave/spatial, fix ave/histo, variable, fix ave/correlate,

Default: none

The option defaults are mode = scalar, ave = one, start = 0, no file output, title 1,2,3 = strings as described above, and no off settings for any input values.

fix bond/break command

Syntax:

```
fix ID group-ID bond/break Nevery bondtype Rmax keyword values ...
```

- ID, group-ID are documented in [fix](#) command
- bond/break = style name of this fix command
- Nevery = attempt bond breaking every this many steps
- bondtype = type of bonds to break
- Rmax = bond longer than Rmax can break (distance units)
- zero or more keyword/value pairs may be appended to args
- keyword = *prob*

```
  prob values = fraction seed
      fraction = break a bond with this probability if otherwise eligible
      seed = random number seed (positive integer)
```

Examples:

```
fix 5 all bond/break 10 2 1.2
fix 5 polymer bond/break 1 1 2.0 prob 0.5 49829
```

Description:

Break bonds between pairs of atoms as a simulation runs according to specified criteria. This can be used to model the dissolution of a polymer network due to stretching of the simulation box or other deformations. In this context, a bond means an interaction between a pair of atoms computed by the [bond_style](#) command. Once the bond is broken it will be permanently deleted. This is different than a [pairwise](#) bond-order potential such as Tersoff or AIREBO which infers bonds and many-body interactions based on the current geometry of a small cluster of atoms and effectively creates and destroys bonds from timestep to timestep as atoms move.

A check for possible bond breakage is performed every *Nevery* timesteps. If two bonded atoms I,J are further than a distance *Rmax* of each other, if the bond is of type *bondtype*, and if both I and J are in the specified fix group, then I,J is labeled as a "possible" bond to break.

If several bonds involving an atom are stretched, it may have multiple possible bonds to break. Every atom checks its list of possible bonds to break and labels the longest such bond as its "sole" bond to break. After this is done, if atom I is bonded to atom J in its sole bond, and atom J is bonded to atom I in its sole bond, then the I,J bond is "eligible" to be broken.

Note that these rules mean an atom will only be part of at most one broken bond on a given timestep. It also means that if atom I chooses atom J as its sole partner, but atom J chooses atom K is its sole partner (due to Rjk > Rij), then this means atom I will not be part of a broken bond on this timestep, even if it has other possible bond partners.

The *prob* keyword can effect whether an eligible bond is actually broken. The *fraction* setting must be a value between 0.0 and 1.0. A uniform random number between 0.0 and 1.0 is generated and the eligible bond is only broken if the random number < fraction.

When a bond is broken, data structures within LIGGGHTS(R)-WSU that store bond topology are updated to reflect the breakage. This can also affect subsequent computation of pairwise interactions involving the atoms

in the bond. See the Restriction section below for additional information.

Computationally, each timestep this fix operates, it loops over bond lists and computes distances between pairs of bonded atoms in the list. It also communicates between neighboring processors to coordinate which bonds are broken. Thus it will increase the cost of a timestep. Thus you should be cautious about invoking this fix too frequently.

You can dump out snapshots of the current bond topology via the dump local command.

IMPORTANT NOTE: Breaking a bond typically alters the energy of a system. You should be careful not to choose bond breaking criteria that induce a dramatic change in energy. For example, if you define a very stiff harmonic bond and break it when 2 atoms are separated by a distance far from the equilibribum bond length, then the 2 atoms will be dramatically released when the bond is broken. More generally, you may need to thermostat your system to compensate for energy changes resulting from broken bonds.

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

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix.

This fix computes two statistics which it stores in a global vector of length 2, which can be accessed by various output commands. The vector values calculated by this fix are "intensive".

These are the 2 quantities:

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

No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions:

This fix is part of the MC package. It is only enabled if LIGGGHTS(R)-WSU was built with that package. See the Making LIGGGHTS(R)-WSU section for more info.

Currently, there are 2 restrictions for using this fix. We may relax these in the future if there are new models that would be enabled by it.

When a bond is broken, you might wish to turn off angle and dihedral interactions that include that bond. However, LIGGGHTS(R)-WSU does not check for these angles and dihedrals, even if your simulation defines an angle_style or dihedral_style.

This fix requires that the pairwise weightings defined by the special_bonds command be 0,1,1 for 1-2, 1-3, and 1-4 neighbors within the bond topology. This effectively means that the pairwise interaction between atoms I and J is turned off when a bond between them exists and will be turned on when the bond is broken. It also means that the pairwise interaction of I with J's other bond partners is unaffected by the existence of the bond.

Related commands:

fix bond/create, fix bond/swap, dump local, special_bonds

Default:

fix bond/break command   178

The option defaults are prob = 1.0.

fix bond/create command

Syntax:

```
fix ID group-ID bond/create Nevery itype jtype Rmin bondtype keyword values ...
```

- ID, group-ID are documented in [fix](#) command
- bond/create = style name of this fix command
- Nevery = attempt bond creation every this many steps
- itype,jtype = atoms of itype can bond to atoms of jtype
- Rmin = 2 atoms separated by less than Rmin can bond (distance units)
- bondtype = type of created bonds
- zero or more keyword/value pairs may be appended to args
- keyword = *iparam* or *jparam* or *prob*

```
  iparam values = maxbond, newtype
     maxbond = max # of bonds of bondtype the itype atom can have
     newtype = change the itype atom to this type when maxbonds exist
   jparam values = maxbond, newtype
     maxbond = max # of bonds of bondtype the jtype atom can have
     newtype = change the jtype atom to this type when maxbonds exist
   prob values = fraction seed
     fraction = create a bond with this probability if otherwise eligible
     seed = random number seed (positive integer)
```

Examples:

```
fix 5 all bond/create 10 1 2 0.8 1
fix 5 all bond/create 1 3 3 0.8 1 prob 0.5 85784 iparam 2 3
```

Description:

Create bonds between pairs of atoms as a simulation runs according to specified criteria. This can be used to model cross-linking of polymers, the formation of a percolation network, etc. In this context, a bond means an interaction between a pair of atoms computed by the [bond_style](#) command. Once the bond is created it will be permanently in place.

A check for possible new bonds is performed every *Nevery* timesteps. If two atoms I,J are within a distance *Rmin* of each other, if I is of atom type *itype*, if J is of atom type *jtype*, if both I and J are in the specified fix group, if a bond does not already exist between I and J, and if both I and J meet their respective *maxbond* requirement (explained below), then I,J is labeled as a "possible" bond pair.

If several atoms are close to an atom, it may have multiple possible bond partners. Every atom checks its list of possible bond partners and labels the closest such partner as its "sole" bond partner. After this is done, if atom I has atom J as its sole partner, and atom J has atom I as its sole partner, then the I,J bond is "eligible" to be formed.

Note that these rules mean an atom will only be part of at most one created bond on a given timestep. It also means that if atom I chooses atom J as its sole partner, but atom J chooses atom K is its sole partner (due to Rjk < Rij), then this means atom I will not form a bond on this timestep, even if it has other possible bond partners.

It is permissible to have *itype* = *jtype*. *Rmin* must be <= the pairwise cutoff distance between *itype* and *jtype* atoms, as defined by the [pair_style](#) command.

The *iparam* and *jparam* keywords can be used to limit the bonding functionality of the participating atoms. Each atom keeps track of how many bonds of *bondtype* it already has. If atom I of itype already has *maxbond* bonds (as set by the *iparam* keyword), then it will not form any more. Likewise for atom J. If *maxbond* is set to 0, then there is no limit on the number of bonds that can be formed with that atom.

The *newtype* value for *iparam* and *jparam* can be used to change the atom type of atom I or J when it reaches *maxbond* number of bonds of type *bondtype*. This means it can now interact in a pairwise fashion with other atoms in a different way by specifying different [pair_coeff](#) coefficients. If you do not wish the atom type to change, simply specify *newtype* as *itype* or *jtype*.

The *prob* keyword can also effect whether an eligible bond is actually created. The *fraction* setting must be a value between 0.0 and 1.0. A uniform random number between 0.0 and 1.0 is generated and the eligible bond is only created if the random number < fraction.

Any bond that is created is assigned a bond type of *bondtype*. Data structures within LIGGGHTS(R)-WSU that store bond topology are updated to reflect the new bond. This can also affect subsequent computation of pairwise interactions involving the atoms in the bond. See the Restriction section below for additional information.

IMPORTANT NOTE: To create a new bond, the internal LIGGGHTS(R)-WSU data structures that store this information must have space for it. When LIGGGHTS(R)-WSU is initialized from a data file, the list of bonds is scanned and the maximum number of bonds per atom is tallied. If some atom will acquire more bonds than this limit as this fix operates, then the "extra bonds per atom" parameter in the data file header must be set to allow for it. See the [read_data](#) command for more details. Note that if this parameter needs to be set, it means a data file must be used to initialize the system, even if it initially has no bonds. A data file with no atoms can be used if you wish to add unbonded atoms via the [create atoms](#) command, e.g. for a percolation simulation.

IMPORTANT NOTE: LIGGGHTS(R)-WSU also maintains a data structure that stores a list of 1st, 2nd, and 3rd neighbors of each atom (in the bond topology of the system) for use in weighting pairwise interactions for bonded atoms. Adding a bond adds a single entry to this list. The "extra" keyword of the [special_bonds](#) command should be used to leave space for new bonds if the maximum number of entries for any atom will be exceeded as this fix operates. See the [special_bonds](#) command for details.

Note that even if your simulation starts with no bonds, you must define a [bond_style](#) and use the [bond_coeff](#) command to specify coefficients for the *bondtype*. Similarly, if new atom types are specified by the *iparam* or *jparam* keywords, they must be within the range of atom types allowed by the simulation and pairwise coefficients must be specified for the new types.

Computationally, each timestep this fix operates, it loops over neighbor lists and computes distances between pairs of atoms in the list. It also communicates between neighboring processors to coordinate which bonds are created. Thus it roughly doubles the cost of a timestep. Thus you should be cautious about invoking this fix too frequently.

You can dump out snapshots of the current bond topology via the [dump local](#) command.

IMPORTANT NOTE: Creating a bond typically alters the energy of a system. You should be careful not to choose bond creation criteria that induce a dramatic change in energy. For example, if you define a very stiff harmonic bond and create it when 2 atoms are separated by a distance far from the equilibribum bond length, then the 2 atoms will oscillate dramatically when the bond is formed. More generally, you may need to thermostat your system to compensate for energy changes resulting from created bonds.

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

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix.

This fix computes two statistics which it stores in a global vector of length 2, which can be accessed by various output commands. The vector values calculated by this fix are "intensive".

These are the 2 quantities:

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

No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions:

This fix is part of the MC package. It is only enabled if LIGGGHTS(R)-WSU was built with that package. See the Making LIGGGHTS(R)-WSU section for more info.

Currently, there are 2 restrictions for using this fix. We may relax these in the future if there are new models that would be enabled by it.

When a bond is created, you might wish to induce new angle and dihedral interactions that include that bond. However, LIGGGHTS(R)-WSU does not create these angles and dihedrals, even if your simulation defines an angle_style or dihedral_style.

This fix requires that the pairwise weightings defined by the special_bonds command be 0,1,1 for 1-2, 1-3, and 1-4 neighbors within the bond topology. This effectively means that the pairwise interaction between atoms I and J will be turned off when a bond between them is created. It also means that the pairwise interaction of I with J's other bond partners will be unaffected by the new bond.

Related commands:

fix bond/break, fix bond/swap, dump local, special_bonds

Default:

The option defaults are iparam = (0,itype), jparam = (0,jtype), and prob = 1.0.

fix box/relax command

Syntax:

```
fix ID group-ID box/relax keyword value ...
```

- ID, group-ID are documented in <u>fix</u> command
- box/relax = style name of this fix command

```
  one or more keyword value pairs may be appended
  keyword = iso or aniso or tri or x or y or z or xy or yz or xz or couple or nreset or vmax
    iso or aniso or tri value = Ptarget = desired pressure (pressure units)
    x or y or z or xy or yz or xz value = Ptarget = desired pressure (pressure units)
    couple = none or xyz or xy or yz or xz
    nreset value = reset reference cell every this many minimizer iterations
    vmax value = fraction = max allowed volume change in one iteration
    dilate value = all or partial
    scaleyz value = yes or no = scale yz with lz
    scalexz value = yes or no = scale xz with lz
    scalexy value = yes or no = scale xy with ly
    fixedpoint values = x y z
      x,y,z = perform relaxation dilation/contraction around this point (distance units)
```

Examples:

```
fix 1 all box/relax iso 0.0 vmax 0.001
fix 2 water box/relax aniso 0.0 dilate partial
fix 2 ice box/relax tri 0.0 couple xy nreset 100
```

Description:

Apply an external pressure or stress tensor to the simulation box during an <u>energy minimization</u>. This allows the box size and shape to vary during the iterations of the minimizer so that the final configuration will be both an energy minimum for the potential energy of the atoms, and the system pressure tensor will be close to the specified external tensor. Conceptually, specifying a positive pressure is like squeezing on the simulation box; a negative pressure typically allows the box to expand.

The external pressure tensor is specified using one or more of the *iso*, *aniso*, *tri*, *x*, *y*, *z*, *xy*, *xz*, *yz*, and *couple* keywords. These keywords give you the ability to specify all 6 components of an external stress tensor, and to couple various of these components together so that the dimensions they represent are varied together during the mimimization.

Orthogonal simulation boxes have 3 adjustable dimensions (x,y,z). Triclinic (non-orthogonal) simulation boxes have 6 adjustable dimensions (x,y,z,xy,xz,yz). The <u>create_box</u>, <u>read data</u>, and <u>read_restart</u> commands specify whether the simulation box is orthogonal or non-orthogonal (triclinic) and explain the meaning of the xy,xz,yz tilt factors.

The target pressures *Ptarget* for each of the 6 components of the stress tensor can be specified independently via the *x*, *y*, *z*, *xy*, *xz*, *yz* keywords, which correspond to the 6 simulation box dimensions. For example, if the *y* keyword is used, the y-box length will change during the minimization. If the *xy* keyword is used, the xy tilt factor will change. A box dimension will not change if that component is not specified.

Note that in order to use the *xy*, *xz*, or *yz* keywords, the simulation box must be triclinic, even if its initial tilt factors are 0.0.

When the size of the simulation box changes, all atoms are re-scaled to new positions, unless the keyword *dilate* is specified with a value of *partial*, in which case only the atoms in the fix group are re-scaled. This can be useful for leaving the coordinates of atoms in a solid substrate unchanged and controlling the pressure of a surrounding fluid.

The *scaleyz*, *scalexz*, and *scalexy* keywords control whether or not the corresponding tilt factors are scaled with the associated box dimensions when relaxing triclinic periodic cells. The default values *yes* will turn on scaling, which corresponds to adjusting the linear dimensions of the cell while preserving its shape. Choosing *no* ensures that the tilt factors are not scaled with the box dimensions. See below for restrictions and default values in different situations. In older versions of LIGGGHTS(R)-WSU, scaling of tilt factors was not performed. The old behavior can be recovered by setting all three scale keywords to *no*.

The *fixedpoint* keyword specifies the fixed point for cell relaxation. By default, it is the center of the box. Whatever point is chosen will not move during the simulation. For example, if the lower periodic boundaries pass through (0,0,0), and this point is provided to *fixedpoint*, then the lower periodic boundaries will remain at (0,0,0), while the upper periodic boundaries will move twice as far. In all cases, the particle positions at each iteration are unaffected by the chosen value, except that all particles are displaced by the same amount, different on each iteration.

IMPORTANT NOTE: Appling an external pressure to tilt dimensions *xy*, *xz*, *yz* can sometimes result in arbitrarily large values of the tilt factors, i.e. a dramatically deformed simulation box. This typically indicates that there is something badly wrong with how the simulation was constructed. The two most common sources of this error are applying a shear stress to a liquid system or specifying an external shear stress tensor that exceeds the yield stress of the solid. In either case the minimization may converge to a bogus conformation or not converge at all. Also note that if the box shape tilts to an extreme shape, LIGGGHTS(R)-WSU will run less efficiently, due to the large volume of communication needed to acquire ghost atoms around a processor's irregular-shaped sub-domain. For extreme values of tilt, LIGGGHTS(R)-WSU may also lose atoms and generate an error.

The *couple* keyword allows two or three of the diagonal components of the pressure tensor to be "coupled" together. The value specified with the keyword determines which are coupled. For example, *xz* means the *Pxx* and *Pzz* components of the stress tensor are coupled. *Xyz* means all 3 diagonal components are coupled. Coupling means two things: the instantaneous stress will be computed as an average of the corresponding diagonal components, and the coupled box dimensions will be changed together in lockstep, meaning coupled dimensions will be dilated or contracted by the same percentage every timestep. The *Ptarget* values for any coupled dimensions must be identical. *Couple xyz* can be used for a 2d simulation; the *z* dimension is simply ignored.

The *iso*, *aniso*, and *tri* keywords are simply shortcuts that are equivalent to specifying several other keywords together.

The keyword *iso* means couple all 3 diagonal components together when pressure is computed (hydrostatic pressure), and dilate/contract the dimensions together. Using "iso Ptarget" is the same as specifying these 4 keywords:

```
x Ptarget
y Ptarget
z Ptarget
couple xyz
```

The keyword *aniso* means *x*, *y*, and *z* dimensions are controlled independently using the *Pxx*, *Pyy*, and *Pzz* components of the stress tensor as the driving forces, and the specified scalar external pressure. Using "aniso Ptarget" is the same as specifying these 4 keywords:

```
x Ptarget
```

```
y Ptarget
z Ptarget
couple none
```

The keyword *tri* means *x*, *y*, *z*, *xy*, *xz*, and *yz* dimensions are controlled independently using their individual stress components as the driving forces, and the specified scalar pressure as the external normal stress. Using "tri Ptarget" is the same as specifying these 7 keywords:

```
x Ptarget
y Ptarget
z Ptarget
xy 0.0
yz 0.0
xz 0.0
couple none
```

The *vmax* keyword can be used to limit the fractional change in the volume of the simulation box that can occur in one iteration of the minimizer. If the pressure is not settling down during the minimization this can be because the volume is fluctuating too much. The specified fraction must be greater than 0.0 and should be << 1.0. A value of 0.001 means the volume cannot change by more than 1/10 of a percent in one iteration when *couple xyz* has been specified. For any other case it means no linear dimension of the simulation box can change by more than 1/10 of a percent.

With this fix, the potential energy used by the minimizer is augmented by an additional energy provided by the fix. The overall objective function then is:

$$E = U + P_t \left(V - V_0 \right) + E_{strain}$$

where *U* is the system potential energy, P_t is the desired hydrostatic pressure, *V* and V_0 are the system and reference volumes, respectively. E_{strain} is the strain energy expression proposed by Parrinello and Rahman (Parrinello1981). Taking derivatives of *E* w.r.t. the box dimensions, and setting these to zero, we find that at the minimum of the objective function, the global system stress tensor **P** will satisfy the relation:

$$\mathbf{P} = P_t \mathbf{I} + \mathbf{S}_t \left(\mathbf{h}_0^{-1} \right)^t \mathbf{h}_{0d}$$

where **I** is the identity matrix, $\mathbf{h}_0$ is the box dimension tensor of the reference cell, and $\mathbf{h}_{0d}$ is the diagonal part of $\mathbf{h}_0$. $\mathbf{S}_t$ is a symmetric stress tensor that is chosen by LIGGGHTS(R)-WSU so that the upper-triangular components of **P** equal the stress tensor specified by the user.

This equation only applies when the box dimensions are equal to those of the reference dimensions. If this is not the case, then the converged stress tensor will not equal that specified by the user. We can resolve this problem by periodically resetting the reference dimensions. The keyword *nreset_ref* controls how often this is done. If this keyword is not used, or is given a value of zero, then the reference dimensions are set to those of the initial simulation domain and are never changed. A value of *nstep* means that every *nstep* minimization steps, the reference dimensions are set to those of the current simulation domain. Note that resetting the reference dimensions changes the objective function and gradients, which sometimes causes the minimization to fail. This can be resolved by changing the value of *nreset*, or simply continuing the minimization from a restart file.

fix box/relax command

IMPORTANT NOTE: As normally computed, pressure includes a kinetic- energy or temperature-dependent component; see the [compute pressure](#) command. However, atom velocities are ignored during a minimization, and the applied pressure(s) specified with this command are assumed to only be the virial component of the pressure (the non-kinetic portion). Thus if atoms have a non-zero temperature and you print the usual thermodynamic pressure, it may not appear the system is converging to your specified pressure. The solution for this is to either (a) zero the velocities of all atoms before performing the minimization, or (b) make sure you are monitoring the pressure without its kinetic component. The latter can be done by outputting the pressure from the fix this command creates (see below) or a pressure fix you define yourself.

IMPORTANT NOTE: Because pressure is often a very sensitive function of volume, it can be difficult for the minimizer to equilibrate the system the desired pressure with high precision, particularly for solids. Some techniques that seem to help are (a) use the "min_modify line quadratic" option when minimizing with box relaxations, and (b) minimize several times in succession if need be, to drive the pressure closer to the target pressure. Also note that some systems (e.g. liquids) will not sustain a non-hydrostatic applied pressure, which means the minimizer will not converge.

This fix computes a temperature and pressure each timestep. The temperature is used to compute the kinetic contribution to the pressure, even though this is subsequently ignored by default. To do this, the fix creates its own computes of style "temp" and "pressure", as if these commands had been issued:

```
compute fix-ID_temp group-ID temp
compute fix-ID_press group-ID pressure fix-ID_temp virial
```

See the [compute temp](#) and [compute pressure](#) commands for details. Note that the IDs of the new computes are the fix-ID + underscore + "temp" or fix_ID + underscore + "press", and the group for the new computes is the same as the fix group. Also note that the pressure compute does not include a kinetic component.

Note that these are NOT the computes used by thermodynamic output (see the [thermo_style](#) command) with ID = *thermo_temp* and *thermo_press*. This means you can change the attributes of this fix's temperature or pressure via the [compute_modify](#) command or print this temperature or pressure during thermodynamic output via the [thermo_style custom](#) command using the appropriate compute-ID. It also means that changing attributes of *thermo_temp* or *thermo_press* will have no effect on this fix.

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

No information about this fix is written to [binary restart files](#).

The [fix_modify](#) *temp* and *press* options are supported by this fix. You can use them to assign a [compute](#) you have defined to this fix which will be used in its temperature and pressure calculation, as described above. Note that as described above, if you assign a pressure compute to this fix that includes a kinetic energy component it will affect the minimization, most likely in an undesirable way.

IMPORTANT NOTE: If both the *temp* and *press* keywords are used in a single thermo_modify command (or in two separate commands), then the order in which the keywords are specified is important. Note that a [pressure compute](#) defines its own temperature compute as an argument when it is specified. The *temp* keyword will override this (for the pressure compute being used by fix npt), but only if the *temp* keyword comes after the *press* keyword. If the *temp* keyword comes before the *press* keyword, then the new pressure compute specified by the *press* keyword will be unaffected by the *temp* setting.

This fix computes a global scalar which can be accessed by various [output commands](#). The scalar is the pressure-volume energy, plus the strain energy, if it exists.

No parameter of this fix can be used with the *start/stop* keywords of the [run](#) command.

This fix is invoked during energy minimization, but not for the purpose of adding a contribution to the energy or forces being minimized. Instead it alters the simulation box geometry as described above.

Restrictions:

Only dimensions that are available can be adjusted by this fix. Non-periodic dimensions are not available. *z*, *xz*, and *yz*, are not available for 2D simulations. *xy*, *xz*, and *yz* are only available if the simulation domain is non-orthogonal. The create_box, read data, and read_restart commands specify whether the simulation box is orthogonal or non-orthogonal (triclinic) and explain the meaning of the xy,xz,yz tilt factors.

The *scaleyz yes* and *scalexz yes* keyword/value pairs can not be used for 2D simulations. *scaleyz yes*, *scalexz yes*, and *scalexy yes* options can only be used if the 2nd dimension in the keyword is periodic, and if the tilt factor is not coupled to the barostat via keywords *tri*, *yz*, *xz*, and *xy*.

Related commands:

fix npt, minimize

Default:

The keyword defaults are dilate = all, vmax = 0.0001, nreset = 0.

(Parrinello1981) Parrinello and Rahman, J Appl Phys, 52, 7182 (1981).

fix buoyancy command

Syntax:

```
fix ID group-ID buoyancy keyword value ...
```

- ID, group-ID are documented in [fix](#) command
- buoyancy = style name of this fix command
- zero or more keyword/value pairs may be appended to args; the *dim* keyword must be used
- keyword = *density* or *dim* or *level* or *region*

```
  density value = density-value
      density-value = fluid density in mass/length^3 units
    dim value = x or y or z
      x, y, z = define the water level as x=const, y=const, z=const plane
    level value = lev
      lev = water level in length units, along the axis defined by dim
    region value = region-ID
      region-ID = ID of region atoms must be in to have added buoyancy force
```

Examples:

```
fix bu all buoyancy level 0.06 dim z density 1000
```

Description:

Add a buoyancy force for each atom in the group. The water level is assumed to be a x=const, y=const or z=const plane, where the axis is defined via the *dim* keyword and const is defined via the *level* keyword. E.g. *dim* = x and *level* = 0.1 would define the water level as x = 0.1.

The buoyancy force is equivalent to the weight of the displaced fluid. Thus, if the particle is fully submerged, the buoyancy force is equal to particle volume * fluid density. If the particle is not submerged, the buoyancy force is 0. If the particle is partially submerged, the fix will calculate the force based on the submerged particle volume. The fluid density is defined by keyword *density*.

As buoyancy is triggered by static pressure difference in a fluid usually caused by gravity, this fix requires to use a [fix gravity](#). The gravity vector has to be specified in the same axis as in this fix (via the *dim* keyword).

If the *region* keyword is used, the atom must also be in the specified geometric [region](#) in order to have buoyancy force added to it.

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

No information about this fix is written to [binary restart files](#). No [fix_modify](#) option applies to this fix. This fix computes a a global 3-vector of the total buoyancy force, which can be accessed by various [output commands](#). No parameter of this fix can be used with the *start/stop* keywords of the [run](#) command.

The forces due to this fix are imposed during an energy minimization, invoked by the [minimize](#) command.

Restrictions:

This fix requires to use a [fix gravity](#). The gravity vector has to be specified in the same axis as in this fix (via the *dim* keyword).

Related commands:

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

Default:

level = 0, *density* = 0

fix check/timestep/gran command

Syntax:

```
fix ID group-ID check/timestep/gran nevery fraction_r fraction_h
```

- ID, group-ID are documented in [fix](#) command
- check/timestep/gran = style name of this fix command
- nevery = evaluate time-step size accuracy every this many time-steps
- fraction_r = warn if time-step size exceeds this fraction of the Rayleigh time
- fraction_h = warn if time-step size exceeds this fraction of the Hertz time

Examples:

```
fix ts_check all check/timestep/gran 1000 0.1 0.1
```

Description:

Periodically calculate estimations of the Rayleigh- and Hertz time dt_r and dt_h for a granular system every 'nevery' time-steps. The user can specify two quantities *fraction_r* and *fraction_h*. A warning message is printed if the time-step size as specified via the [timestep](#) command exceeds either of dt_r * fraction_r or dt_h * fraction_h.

The former quantity is

$dt_r = PI*r*sqrt(rho/G)/(0.1631*nu+0.8766)$,

where rho is particle density, G is the shear modulus and nu is Poisson's ratio. The latter quantity is expressed by

$dt_h = 2.87*(m\_eff^2/(r\_eff*Y\_eff^2*v\_max))^{0.2}$.

The effective mass m_eff, the effective radius r_eff and the effective Young's modulus Y_eff are as defined in [pair gran](#). v_max is the maximum relative velocity, taking mesh movement into account. Please note that the Hertz criterion will also be used if you use a different granular pair style (e.g. Hooke).

Additionally, this command checks the ratio of skin to the distance that particles can travel relative to each other in one time-step. This value should be >1, otherwise some interactions may be missed or overlap energy may be generated artificially. This command will warn you if this is the case.

These criteria are checked every 'nevery' time-steps. Rayleigh time dt_r is calculated for each particle in the simulation, and the minimum value is taken for further calculations. Hertz time dt_h is estimated by testing a collision of each particle with itself using v_max as the assumed collision velocity.

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

No information about this fix is written to [binary restart files](#). None of the [fix_modify](#) options are relevant to this fix. This fix computes a 3-vector, for access by various [output commands](#). The vector consists of the time-step size expressed as fraction of the Rayleigh and Hertz time-step sizes and the ratio of skin to the distance particles can travel relative to each other in one time-step, respectively. No parameter of this fix can be used with the *start/stop* keywords of the [run](#) command. This fix is not invoked during [energy minimization](#).

Restrictions: none

Related commands: none

Default: none

fix deform command

Syntax:

```
fix ID group-ID deform N parameter args ... keyword value ...
```

- ID, group-ID are documented in fix command
- deform = style name of this fix command
- N = perform box deformation every this many timesteps
- one or more parameter/arg pairs may be appended

```
parameter = x or y or z or xy or xz or yz
  x, y, z args = style value(s)
    style = final or delta or scale or vel or erate or trate or volume or wiggle or variab
      final values = lo hi
        lo hi = box boundaries at end of run (distance units)
      delta values = dlo dhi
        dlo dhi = change in box boundaries at end of run (distance units)
      scale values = factor
        factor = multiplicative factor for change in box length at end of run
      vel value = V
        V = change box length at this velocity (distance/time units),
            effectively an engineering strain rate
      erate value = R
        R = engineering strain rate (1/time units)
      trate value = R
        R = true strain rate (1/time units)
      volume value = none = adjust this dim to preserve volume of system
      wiggle values = A Tp
        A = amplitude of oscillation (distance units)
        Tp = period of oscillation (time units)
      variable values = v_name1 v_name2
        v_name1 = variable with name1 for box length change as function of time
        v_name2 = variable with name2 for change rate as function of time
  xy, xz, yz args = style value
    style = final or delta or vel or erate or trate or wiggle
      final value = tilt
        tilt = tilt factor at end of run (distance units)
      delta value = dtilt
        dtilt = change in tilt factor at end of run (distance units)
      vel value = V
        V = change tilt factor at this velocity (distance/time units),
            effectively an engineering shear strain rate
      erate value = R
        R = engineering shear strain rate (1/time units)
      trate value = R
        R = true shear strain rate (1/time units)
      wiggle values = A Tp
        A = amplitude of oscillation (distance units)
        Tp = period of oscillation (time units)
      variable values = v_name1 v_name2
        v_name1 = variable with name1 for tilt change as function of time
        v_name2 = variable with name2 for change rate as function of time
```

- zero or more keyword/value pairs may be appended
- keyword = *remap* or *flip* or *units*

```
remap value = x or v or none
    x = remap coords of atoms in group into deforming box
    v = remap velocities of all atoms when they cross periodic boundaries
    none = no remapping of x or v
```

```
    flip value = yes or no
      allow or disallow box flips when it becomes highly skewed
    units value = lattice or box
      lattice = distances are defined in lattice units
      box = distances are defined in simulation box units
```

Examples:

```
fix 1 all deform 1 x final 0.0 9.0 z final 0.0 5.0 units box
fix 1 all deform 1 x trate 0.1 y volume z volume
fix 1 all deform 1 xy erate 0.001 remap v
fix 1 all deform 10 y delta -0.5 0.5 xz vel 1.0
```

Description:

Change the volume and/or shape of the simulation box during a dynamics run. Orthogonal simulation boxes have 3 adjustable parameters (x,y,z). Triclinic (non-orthogonal) simulation boxes have 6 adjustable parameters (x,y,z,xy,xz,yz). Any or all of them can be adjusted independently and simultaneously by this command.

For the *x*, *y*, *z* parameters, the associated dimension cannot be shrink-wrapped. For the *xy*, *yz*, *xz* parameters, the associated 2nd dimension cannot be shrink-wrapped. Dimensions not varied by this command can be periodic or non-periodic. Dimensions corresponding to unspecified parameters can also be controlled by a fix npt or fix nph command.

The size and shape of the simulation box at the beginning of the simulation run were either specified by the create_box or read_data or read_restart command used to setup the simulation initially if it is the first run, or they are the values from the end of the previous run. The create_box, read data, and read_restart commands specify whether the simulation box is orthogonal or non-orthogonal (triclinic) and explain the meaning of the xy,xz,yz tilt factors. If fix deform changes the xy,xz,yz tilt factors, then the simulation box must be triclinic, even if its initial tilt factors are 0.0.

As described below, the desired simulation box size and shape at the end of the run are determined by the parameters of the fix deform command. Every Nth timestep during the run, the simulation box is expanded, contracted, or tilted to ramped values between the initial and final values.

For the *x*, *y*, and *z* parameters, this is the meaning of their styles and values.

The *final*, *delta*, *scale*, *vel*, and *erate* styles all change the specified dimension of the box via "constant displacement" which is effectively a "constant engineering strain rate". This means the box dimension changes linearly with time from its initial to final value.

For style *final*, the final lo and hi box boundaries of a dimension are specified. The values can be in lattice or box distance units. See the discussion of the units keyword below.

For style *delta*, plus or minus changes in the lo/hi box boundaries of a dimension are specified. The values can be in lattice or box distance units. See the discussion of the units keyword below.

For style *scale*, a multiplicative factor to apply to the box length of a dimension is specified. For example, if the initial box length is 10, and the factor is 1.1, then the final box length will be 11. A factor less than 1.0 means compression.

For style *vel*, a velocity at which the box length changes is specified in units of distance/time. This is effectively a "constant engineering strain rate", where rate = V/L0 and L0 is the initial box length. The distance can be in lattice or box distance units. See the discussion of the units keyword below. For example, if the initial box length is 100 Angstroms, and V is 10 Angstroms/psec, then after 10 psec, the box length will

have doubled. After 20 psec, it will have tripled.

The *erate* style changes a dimension of the the box at a "constant engineering strain rate". The units of the specified strain rate are 1/time. See the [units](#) command for the time units associated with different choices of simulation units, e.g. picoseconds for "metal" units). Tensile strain is unitless and is defined as delta/L0, where L0 is the original box length and delta is the change relative to the original length. The box length L as a function of time will change as

```
L(t) = L0 (1 + erate*dt)
```

where dt is the elapsed time (in time units). Thus if *erate* R is specified as 0.1 and time units are picoseconds, this means the box length will increase by 10% of its original length every picosecond. I.e. strain after 1 psec = 0.1, strain after 2 psec = 0.2, etc. R = -0.01 means the box length will shrink by 1% of its original length every picosecond. Note that for an "engineering" rate the change is based on the original box length, so running with R = 1 for 10 picoseconds expands the box length by a factor of 11 (strain of 10), which is different that what the *trate* style would induce.

The *trate* style changes a dimension of the box at a "constant true strain rate". Note that this is not an "engineering strain rate", as the other styles are. Rather, for a "true" rate, the rate of change is constant, which means the box dimension changes non-linearly with time from its initial to final value. The units of the specified strain rate are 1/time. See the [units](#) command for the time units associated with different choices of simulation units, e.g. picoseconds for "metal" units). Tensile strain is unitless and is defined as delta/L0, where L0 is the original box length and delta is the change relative to the original length.

The box length L as a function of time will change as

```
L(t) = L0 exp(trate*dt)
```

where dt is the elapsed time (in time units). Thus if *trate* R is specified as ln(1.1) and time units are picoseconds, this means the box length will increase by 10% of its current (not original) length every picosecond. I.e. strain after 1 psec = 0.1, strain after 2 psec = 0.21, etc. R = ln(2) or ln(3) means the box length will double or triple every picosecond. R = ln(0.99) means the box length will shrink by 1% of its current length every picosecond. Note that for a "true" rate the change is continuous and based on the current length, so running with R = ln(2) for 10 picoseconds does not expand the box length by a factor of 11 as it would with *erate*, but by a factor of 1024 since the box length will double every picosecond.

Note that to change the volume (or cross-sectional area) of the simulation box at a constant rate, you can change multiple dimensions via *erate* or *trate*. E.g. to double the box volume in a picosecond picosecond, you could set "x erate M", "y erate M", "z erate M", with M = pow(2,1/3) - 1 = 0.26, since if each box dimension grows by 26%, the box volume doubles. Or you could set "x trate M", "y trate M", "z trate M", with M = ln(1.26) = 0.231, and the box volume would double every picosecond.

The *volume* style changes the specified dimension in such a way that the box volume remains constant while other box dimensions are changed explicitly via the styles discussed above. For example, "x scale 1.1 y scale 1.1 z volume" will shrink the z box length as the x,y box lengths increase, to keep the volume constant (product of x,y,z lengths). If "x scale 1.1 z volume" is specified and parameter *y* is unspecified, then the z box length will shrink as x increases to keep the product of x,z lengths constant. If "x scale 1.1 y volume z volume" is specified, then both the y,z box lengths will shrink as x increases to keep the volume constant (product of x,y,z lengths). In this case, the y,z box lengths shrink so as to keep their relative aspect ratio constant.

For solids or liquids, note that when one dimension of the box is expanded via fix deform (i.e. tensile strain), it may be physically undesirable to hold the other 2 box lengths constant (unspecified by fix deform) since that implies a density change. Using the *volume* style for those 2 dimensions to keep the box volume constant may make more physical sense, but may also not be correct for materials and potentials whose Poisson ratio is

fix deform command

not 0.5. An alternative is to use fix npt aniso with zero applied pressure on those 2 dimensions, so that they respond to the tensile strain dynamically.

The *wiggle* style oscillates the specified box length dimension sinusoidally with the specified amplitude and period. I.e. the box length L as a function of time is given by

```
L(t) = L0 + A sin(2*pi t/Tp)
```

where L0 is its initial length. If the amplitude A is a positive number the box initially expands, then contracts, etc. If A is negative then the box initially contracts, then expands, etc. The amplitude can be in lattice or box distance units. See the discussion of the units keyword below.

The *variable* style changes the specified box length dimension by evaluating a variable, which presumably is a function of time. The variable with *name1* must be an equal-style variable and should calculate a change in box length in units of distance. Note that this distance is in box units, not lattice units; see the discussion of the *units* keyword below. The formula associated with variable *name1* can reference the current timestep. Note that it should return the "change" in box length, not the absolute box length. This means it should evaluate to 0.0 when invoked on the initial timestep of the run following the definition of fix deform. It should evaluate to a value > 0.0 to dilate the box at future times, or a value < 0.0 to compress the box.

The variable *name2* must also be an equal-style variable and should calculate the rate of box length change, in units of distance/time, i.e. the time-derivative of the *name1* variable. This quantity is used internally by LIGGGHTS(R)-WSU to reset atom velocities when they cross periodic boundaries. It is computed internally for the other styles, but you must provide it when using an arbitrary variable.

Here is an example of using the *variable* style to perform the same box deformation as the *wiggle* style formula listed above, where we assume that the current timestep = 0.

```
variable A equal 5.0
variable Tp equal 10.0
variable displace equal "v_A * sin(2*PI * step*dt/v_Tp)"
variable rate equal "2*PI*v_A/v_Tp * cos(2*PI * step*dt/v_Tp)"
fix 2 all deform 1 x variable v_displace v_rate remap v
```

For the *scale*, *vel*, *erate*, *trate*, *volume*, *wiggle*, and *variable* styles, the box length is expanded or compressed around its mid point.

For the *xy*, *xz*, and *yz* parameters, this is the meaning of their styles and values. Note that changing the tilt factors of a triclinic box does not change its volume.

The *final*, *delta*, *vel*, and *erate* styles all change the shear strain at a "constant engineering shear strain rate". This means the tilt factor changes linearly with time from its initial to final value.

For style *final*, the final tilt factor is specified. The value can be in lattice or box distance units. See the discussion of the units keyword below.

For style *delta*, a plus or minus change in the tilt factor is specified. The value can be in lattice or box distance units. See the discussion of the units keyword below.

For style *vel*, a velocity at which the tilt factor changes is specified in units of distance/time. This is effectively an "engineering shear strain rate", where rate = V/L0 and L0 is the initial box length perpendicular to the direction of shear. The distance can be in lattice or box distance units. See the discussion of the units keyword below. For example, if the initial tilt factor is 5 Angstroms, and the V is 10 Angstroms/psec, then after 1 psec, the tilt factor will be 15 Angstroms. After 2 psec, it will be 25 Angstroms.

The *erate* style changes a tilt factor at a "constant engineering shear strain rate". The units of the specified shear strain rate are 1/time. See the [units](#) command for the time units associated with different choices of simulation units, e.g. picoseconds for "metal" units). Shear strain is unitless and is defined as offset/length, where length is the box length perpendicular to the shear direction (e.g. y box length for xy deformation) and offset is the displacement distance in the shear direction (e.g. x direction for xy deformation) from the unstrained orientation.

The tilt factor T as a function of time will change as

```
T(t) = T0 + L0*erate*dt
```

where T0 is the initial tilt factor, L0 is the original length of the box perpendicular to the shear direction (e.g. y box length for xy deformation), and dt is the elapsed time (in time units). Thus if *erate* R is specified as 0.1 and time units are picoseconds, this means the shear strain will increase by 0.1 every picosecond. I.e. if the xy shear strain was initially 0.0, then strain after 1 psec = 0.1, strain after 2 psec = 0.2, etc. Thus the tilt factor would be 0.0 at time 0, 0.1*ybox at 1 psec, 0.2*ybox at 2 psec, etc, where ybox is the original y box length. R = 1 or 2 means the tilt factor will increase by 1 or 2 every picosecond. R = -0.01 means a decrease in shear strain by 0.01 every picosecond.

The *trate* style changes a tilt factor at a "constant true shear strain rate". Note that this is not an "engineering shear strain rate", as the other styles are. Rather, for a "true" rate, the rate of change is constant, which means the tilt factor changes non-linearly with time from its initial to final value. The units of the specified shear strain rate are 1/time. See the [units](#) command for the time units associated with different choices of simulation units, e.g. picoseconds for "metal" units). Shear strain is unitless and is defined as offset/length, where length is the box length perpendicular to the shear direction (e.g. y box length for xy deformation) and offset is the displacement distance in the shear direction (e.g. x direction for xy deformation) from the unstrained orientation.

The tilt factor T as a function of time will change as

```
T(t) = T0 exp(trate*dt)
```

where T0 is the initial tilt factor and dt is the elapsed time (in time units). Thus if *trate* R is specified as ln(1.1) and time units are picoseconds, this means the shear strain or tilt factor will increase by 10% every picosecond. I.e. if the xy shear strain was initially 0.1, then strain after 1 psec = 0.11, strain after 2 psec = 0.121, etc. R = ln(2) or ln(3) means the tilt factor will double or triple every picosecond. R = ln(0.99) means the tilt factor will shrink by 1% every picosecond. Note that the change is continuous, so running with R = ln(2) for 10 picoseconds does not change the tilt factor by a factor of 10, but by a factor of 1024 since it doubles every picosecond. Note that the initial tilt factor must be non-zero to use the *trate* option.

Note that shear strain is defined as the tilt factor divided by the perpendicular box length. The *erate* and *trate* styles control the tilt factor, but assume the perpendicular box length remains constant. If this is not the case (e.g. it changes due to another fix deform parameter), then this effect on the shear strain is ignored.

The *wiggle* style oscillates the specified tilt factor sinusoidally with the specified amplitude and period. I.e. the tilt factor T as a function of time is given by

```
T(t) = T0 + A sin(2*pi t/Tp)
```

where T0 is its initial value. If the amplitude A is a positive number the tilt factor initially becomes more positive, then more negative, etc. If A is negative then the tilt factor initially becomes more negative, then more positive, etc. The amplitude can be in lattice or box distance units. See the discussion of the units keyword below.

The *variable* style changes the specified tilt factor by evaluating a variable, which presumably is a function of time. The variable with *name1* must be an [equal-style variable](#) and should calculate a change in tilt in units of distance. Note that this distance is in box units, not lattice units; see the discussion of the *units* keyword below. The formula associated with variable *name1* can reference the current timestep. Note that it should return the "change" in tilt factor, not the absolute tilt factor. This means it should evaluate to 0.0 when invoked on the initial timestep of the run following the definition of fix deform.

The variable *name2* must also be an [equal-style variable](#) and should calculate the rate of tilt change, in units of distance/time, i.e. the time-derivative of the *name1* variable. This quantity is used internally by LIGGGHTS(R)-WSU to reset atom velocities when they cross periodic boundaries. It is computed internally for the other styles, but you must provide it when using an arbitrary variable.

Here is an example of using the *variable* style to perform the same box deformation as the *wiggle* style formula listed above, where we assume that the current timestep = 0.

```
variable A equal 5.0
variable Tp equal 10.0
variable displace equal "v_A * sin(2*PI * step*dt/v_Tp)"
variable rate equal "2*PI*v_A/v_Tp * cos(2*PI * step*dt/v_Tp)"
fix 2 all deform 1 xy variable v_displace v_rate remap v
```

All of the tilt styles change the xy, xz, yz tilt factors during a simulation. In LIGGGHTS(R)-WSU, tilt factors (xy,xz,yz) for triclinic boxes are normally bounded by half the distance of the parallel box length. See the discussion of the *flip* keyword below, to allow this bound to be exceeded, if desired.

For example, if xlo = 2 and xhi = 12, then the x box length is 10 and the xy tilt factor must be between -5 and 5. Similarly, both xz and yz must be between -(xhi-xlo)/2 and +(yhi-ylo)/2. Note that this is not a limitation, since if the maximum tilt factor is 5 (as in this example), then configurations with tilt = ..., -15, -5, 5, 15, 25, ... are all equivalent.

To obey this constraint and allow for large shear deformations to be applied via the *xy*, *xz*, or *yz* parameters, the following algorithm is used. If *prd* is the associated parallel box length (10 in the example above), then if the tilt factor exceeds the accepted range of -5 to 5 during the simulation, then the box is flipped to the other limit (an equivalent box) and the simulation continues. Thus for this example, if the initial xy tilt factor was 0.0 and "xy final 100.0" was specified, then during the simulation the xy tilt factor would increase from 0.0 to 5.0, the box would be flipped so that the tilt factor becomes -5.0, the tilt factor would increase from -5.0 to 5.0, the box would be flipped again, etc. The flip occurs 10 times and the final tilt factor at the end of the simulation would be 0.0. During each flip event, atoms are remapped into the new box in the appropriate manner.

The one exception to this rule is if the 1st dimension in the tilt factor (x for xy) is non-periodic. In that case, the limits on the tilt factor are not enforced, since flipping the box in that dimension does not change the atom positions due to non-periodicity. In this mode, if you tilt the system to extreme angles, the simulation will simply become inefficient due to the highly skewed simulation box.

Each time the box size or shape is changed, the *remap* keyword determines whether atom positions are remapped to the new box. If *remap* is set to *x* (the default), atoms in the fix group are remapped; otherwise they are not. Note that their velocities are not changed, just their positions are altered. If *remap* is set to *v*, then any atom in the fix group that crosses a periodic boundary will have a delta added to its velocity equal to the difference in velocities between the lo and hi boundaries. Note that this velocity difference can include tilt components, e.g. a delta in the x velocity when an atom crosses the y periodic boundary. If *remap* is set to *none*, then neither of these remappings take place.

Conceptually, setting *remap* to *x* forces the atoms to deform via an affine transformation that exactly matches the box deformation. This setting is typically appropriate for solids. Note that though the atoms are effectively

"moving" with the box over time, it is not due to their having a velocity that tracks the box change, but only due to the remapping. By contrast, setting *remap* to *v* is typically appropriate for fluids, where you want the atoms to respond to the change in box size/shape on their own and acquire a velocity that matches the box change, so that their motion will naturally track the box without explicit remapping of their coordinates.

IMPORTANT NOTE: When non-equilibrium MD (NEMD) simulations are performed using this fix, the option "remap v" should normally be used. This is because fix nvt/sllod adjusts the atom positions and velocities to induce a velocity profile that matches the changing box size/shape. Thus atom coordinates should NOT be remapped by fix deform, but velocities SHOULD be when atoms cross periodic boundaries, since that is consistent with maintaining the velocity profile already created by fix nvt/sllod. LIGGGHTS(R)-WSU will warn you if the *remap* setting is not consistent with fix nvt/sllod.

IMPORTANT NOTE: For non-equilibrium MD (NEMD) simulations using "remap v" it is usually desirable that the fluid (or flowing material, e.g. granular particles) stream with a velocity profile consistent with the deforming box. As mentioned above, using a thermostat such as fix nvt/sllod or fix lavgevin (with a bias provided by compute temp/deform), will typically accomplish that. If you do not use a thermostat, then there is no driving force pushing the atoms to flow in a manner consistent with the deforming box. E.g. for a shearing system the box deformation velocity may vary from 0 at the bottom to 10 at the top of the box. But the stream velocity profile of the atoms may vary from -5 at the bottom to +5 at the top. You can monitor these effects using the fix ave/spatial, compute temp/deform, and compute temp/profile commands. One way to induce atoms to stream consistent with the box deformation is to give them an initial velocity profile, via the velocity ramp command, that matches the box deformation rate. This also typically helps the system come to equilibrium more quickly, even if a thermostat is used.

IMPORTANT NOTE: If a fix rigid is defined for rigid bodies, and *remap* is set to *x*, then the center-of-mass coordinates of rigid bodies will be remapped to the changing simulation box. This will be done regardless of whether atoms in the rigid bodies are in the fix deform group or not. The velocity of the centers of mass are not remapped even if *remap* is set to *v*, since fix nvt/sllod does not currently do anything special for rigid particles. If you wish to perform a NEMD simulation of rigid particles, you can either thermostat them independently or include a background fluid and thermostat the fluid via fix nvt/sllod.

The *flip* keyword allows the tilt factors for a triclinic box to exceed half the distance of the parallel box length, as discussed above. If the *flip* value is set to *yes*, the bound is enforced by flipping the box when it is exceeded. If the *flip* value is set to *no*, the tilt will continue to change without flipping. Note that if you apply large deformations, this means the box shape can tilt dramatically LIGGGHTS(R)-WSU will run less efficiently, due to the large volume of communication needed to acquire ghost atoms around a processor's irregular-shaped sub-domain. For extreme values of tilt, LIGGGHTS(R)-WSU may also lose atoms and generate an error.

The *units* keyword determines the meaning of the distance units used to define various arguments. A *box* value selects standard distance units as defined by the units command, e.g. Angstroms for units = real or metal. A *lattice* value means the distance units are in lattice spacings. The lattice command must have been previously used to define the lattice spacing. Note that the units choice also affects the *vel* style parameters since it is defined in terms of distance/time. Also note that the units keyword does not affect the *variable* style. You should use the *xlat*, *ylat*, *zlat* keywords of the thermo_style command if you want to include lattice spacings in a variable formula.

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

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix. No global or per-atom quantities are stored by this fix for access by various output commands.

This fix can perform deformation over multiple runs, using the *start* and *stop* keywords of the run command. See the run command for details of how to do this.

This fix is not invoked during [energy minimization](#).

Restrictions:

You cannot apply x, y, or z deformations to a dimension that is shrink-wrapped via the [boundary](#) comamnd.

You cannot apply xy, yz, or xz deformations to a 2nd dimension (y in xy) that is shrink-wrapped via the [boundary](#) comamnd.

Related commands:

[change_box](#)

Default:

The option defaults are remap = x, flip = yes, and units = box.

fix drag command

Syntax:

```
fix ID group-ID drag x y z fmag delta
```

- ID, group-ID are documented in [fix](#) command
- drag = style name of this fix command
- x,y,z = coord to drag atoms towards
- fmag = magnitude of force to apply to each atom (force units)
- delta = cutoff distance inside of which force is not applied (distance units)

Examples:

```
fix center small-molecule drag 0.0 10.0 0.0 5.0 2.0
```

Description:

Apply a force to each atom in a group to drag it towards the point (x,y,z). The magnitude of the force is specified by fmag. If an atom is closer than a distance delta to the point, then the force is not applied.

Any of the x,y,z values can be specified as NULL which means do not include that dimension in the distance calculation or force application.

This command can be used to steer one or more atoms to a new location in the simulation.

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

No information about this fix is written to [binary restart files](#). None of the [fix_modify](#) options are relevant to this fix.

This fix computes a global 3-vector of forces, which can be accessed by various [output commands](#). This is the total force on the group of atoms by the drag force. The vector values calculated by this fix are "extensive".

No parameter of this fix can be used with the *start/stop* keywords of the [run](#) command. This fix is not invoked during [energy minimization](#).

Restrictions: none

Related commands:

[fix spring](#), [fix spring/self](#), [fix spring/rg](#), [fix smd](#)

Default: none

fix dt/reset command

Syntax:

```
fix ID group-ID dt/reset N Tmin Tmax Xmax keyword values ...
```

- ID, group-ID are documented in [fix](#) command
- dt/reset = style name of this fix command
- N = recompute dt every N timesteps
- Tmin = minimum dt allowed which can be NULL (time units)
- Tmax = maximum dt allowed which can be NULL (time units)
- Xmax = maximum distance for an atom to move in one timestep (distance units)
- zero or more keyword/value pairs may be appended
- keyword = *units*

```
units value = lattice or box
    lattice = Xmax is defined in lattice units
    box = Xmax is defined in simulation box units
```

Examples:

```
fix 5 all dt/reset 10 1.0e-5 0.01 0.1
fix 5 all dt/reset 10 0.01 2.0 0.2 units box
```

Description:

Reset the timestep size every N steps during a run, so that no atom moves further than Xmax, based on current atom velocities and forces. This can be useful when starting from a configuration with overlapping atoms, where forces will be large. Or it can be useful when running an impact simulation where one or more high-energy atoms collide with a solid, causing a damage cascade.

This fix overrides the timestep size setting made by the [timestep](#) command. The new timestep size *dt* is computed in the following manner.

For each atom, the timestep is computed that would cause it to displace *Xmax* on the next integration step, as a function of its current velocity and force. Since performing this calculation exactly would require the solution to a quartic equation, a cheaper estimate is generated. The estimate is conservative in that the atom's displacement is guaranteed not to exceed *Xmax*, though it may be smaller.

Given this putative timestep for each atom, the minimum timestep value across all atoms is computed. Then the *Tmin* and *Tmax* bounds are applied, if specified. If one (or both) is specified as NULL, it is not applied.

When the [run style](#) is *respa*, this fix resets the outer loop (largest) timestep, which is the same timestep that the [timestep](#) command sets.

Note that the cumulative simulation time (in time units), which accounts for changes in the timestep size as a simulation proceeds, can be accessed by the [thermo_style time](#) keyword.

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

No information about this fix is written to [binary restart files](#). None of the [fix_modify](#) options are relevant to this fix.

This fix computes a global scalar which can be accessed by various [output commands](). The scalar stores the last timestep on which the timestep was reset to a new value.

The scalar value calculated by this fix is "intensive".

No parameter of this fix can be used with the *start/stop* keywords of the [run]() command. This fix is not invoked during [energy minimization]().

Restrictions: none

Related commands:

[timestep]()

Default:

The option defaults is units = box.

fix efield command

Syntax:

```
fix ID group-ID efield ex ey ez keyword value ...
```

- ID, group-ID are documented in fix command
- efield = style name of this fix command
- ex,ey,ez = E-field component values (electric field units)
- any of ex,ey,ez can be a variable (see below)
- zero or more keyword/value pairs may be appended to args
- keyword = *region* or *energy*

```
  region value = region-ID
      region-ID = ID of region atoms must be in to have added force
    energy value = v_name
      v_name = variable with name that calculates the potential energy of each atom in the a
```

Examples:

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

Description:

Add a force F = qE to each charged atom in the group due to an external electric field being applied to the system. If the system contains point-dipoles, also add a torque on the dipoles due to the external electric field.

For charges, any of the 3 quantities defining the E-field components can be specified as an equal-style or atom-style variable, namely *ex*, *ey*, *ez*. If the value is a variable, it should be specified as v_name, where name is the variable name. In this case, the variable will be evaluated each timestep, and its value used to determine the E-field component.

For point-dipoles, equal-style variables can be used, but atom-style variables are not currently supported, since they imply a spatial gradient in the electric field which means additional terms with gradients of the field are required for the force and torque on dipoles.

Equal-style variables can specify formulas with various mathematical functions, and include thermo_style command keywords for the simulation box parameters and timestep and elapsed time. Thus it is easy to specify a time-dependent E-field.

Atom-style variables can specify the same formulas as equal-style variables but can also include per-atom values, such as atom coordinates. Thus it is easy to specify a spatially-dependent E-field with optional time-dependence as well.

If the *region* keyword is used, the atom must also be in the specified geometric region in order to have force added to it.

Adding a force or torque to atoms implies a change in their potential energy as they move or rotate due to the applied E-field.

For dynamics via the "run" command, this energy can be optionally added to the system's potential energy for thermodynamic output (see below). For energy minimization via the "minimize" command, this energy must be added to the system's potential energy to formulate a self-consistent minimization problem (see below).

The *energy* keyword is not allowed if the added field is a constant vector (ex,ey,ez), with all components defined as numeric constants and not as variables. This is because LIGGGHTS(R)-WSU can compute the energy for each charged particle directly as E = -x dot qE = -q (x*ex + y*ey + z*ez), so that -Grad(E) = F. Similarly for point-dipole particles the energy can be computed as E = -mu dot E = -(mux*ex + muy*ey + muz*ez).

The *energy* keyword is optional if the added force is defined with one or more variables, and if you are performing dynamics via the run command. If the keyword is not used, LIGGGHTS(R)-WSU will set the energy to 0.0, which is typically fine for dynamics.

The *energy* keyword is required if the added force is defined with one or more variables, and you are performing energy minimization via the "minimize" command for charged particles. It is not required for point-dipoles, but a warning is issued since the minimizer in LIGGGHTS(R)-WSU does not rotate dipoles, so you should not expect to be able to minimize the orientation of dipoles in an applied electric field.

The *energy* keyword specifies the name of an atom-style variable which is used to compute the energy of each atom as function of its position. Like variables used for *ex*, *ey*, *ez*, the energy variable is specified as v_name, where name is the variable name.

Note that when the *energy* keyword is used during an energy minimization, you must insure that the formula defined for the atom-style variable is consistent with the force variable formulas, i.e. that -Grad(E) = F. For example, if the force due to the electric field were a spring-like F = kx, then the energy formula should be E = -0.5kx^2. If you don't do this correctly, the minimization will not converge properly.

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

No information about this fix is written to binary restart files.

The fix_modify *energy* option is supported by this fix to add the potential "energy" inferred by the added force due to the electric field to the system's potential energy as part of thermodynamic output. This is a fictitious quantity but is needed so that the minimize command can include the forces added by this fix in a consistent manner. I.e. there is a decrease in potential energy when atoms move in the direction of the added force due to the electric field.

This fix computes a global scalar and a global 3-vector of forces, which can be accessed by various output commands. The scalar is the potential energy discussed above. The vector is the total force added to the group of atoms. The scalar and vector values calculated by this fix are "extensive".

No parameter of this fix can be used with the *start/stop* keywords of the run command.

The forces due to this fix are imposed during an energy minimization, invoked by the minimize command. You should not specify force components with a variable that has time-dependence for use with a minimizer, since the minimizer increments the timestep as the iteration count during the minimization.

IMPORTANT NOTE: If you want the fictitious potential energy associated with the added forces to be included in the total potential energy of the system (the quantity being minimized), you MUST enable the fix_modify *energy* option for this fix.

Restrictions: none

Related commands:

fix addforce

Default: none

fix enforce2d command

Syntax:

```
fix ID group-ID enforce2d
```

- ID, group-ID are documented in [fix](#) command
- enforce2d = style name of this fix command

Examples:

```
fix 5 all enforce2d
```

Description:

Zero out the z-dimension velocity and force on each atom in the group. This is useful when running a 2d simulation to insure that atoms do not move from their initial z coordinate.

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

No information about this fix is written to [binary restart files](#). None of the [fix_modify](#) options are relevant to this fix. No global or per-atom quantities are stored by this fix for access by various [output commands](#). No parameter of this fix can be used with the *start/stop* keywords of the [run](#) command.

The forces due to this fix are imposed during an energy minimization, invoked by the [minimize](#) command.

Restrictions: none

Related commands: none

Default: none

fix external command

Syntax:

```
fix ID group-ID external mode args
```

- ID, group-ID are documented in fix command
- external = style name of this fix command
- mode = *pf/callback* or *pf/array*

```
  pf/callback args = Ncall Napply
      Ncall = make callback every Ncall steps
      Napply = apply callback forces every Napply steps
    pf/array args = Napply
      Napply = apply array forces every Napply steps
```

Examples:

```
fix 1 all external pf/callback 1 1
fix 1 all external pf/callback 100 1
fix 1 all external pf/array 10
```

Description:

This fix allows external programs that are running LIGGGHTS(R)-WSU through its library interface to modify certain LIGGGHTS(R)-WSU properties on specific timesteps, similar to the way other fixes do. The external driver can be a C/C++ or Fortran program or a Python script.

If mode is *pf/callback* then the fix will make a callback every *Ncall* timesteps or minimization iterations to the external program. The external program computes forces on atoms by setting values in an array owned by the fix. The fix then adds these forces to each atom in the group, once every *Napply* steps, similar to the way the fix addforce command works. Note that if *Ncall* > *Napply*, the force values produced by one callback will persist, and be used multiple times to update atom forces.

The callback function "foo" is invoked by the fix as:

```
foo(void *ptr, bigint timestep, int nlocal, int *ids, double **x, double **fexternal);
```

The arguments are as follows:

- ptr = pointer provided by and simply passed back to external driver
- timestep = current LIGGGHTS(R)-WSU timestep
- nlocal = # of atoms on this processor
- ids = list of atom IDs on this processor
- x = coordinates of atoms on this processor
- fexternal = forces to add to atoms on this processor

Note that timestep is a "bigint" which is defined in src/lmptype.h, typically as a 64-bit integer.

Fexternal are the forces returned by the driver program.

The fix has a set_callback() method which the external driver can call to pass a pointer to its foo() function. See the couple/lammps_quest/lmpqst.cpp file in the LIGGGHTS(R)-WSU distribution for an example of how

this is done. This sample application performs classical MD using quantum forces computed by a density functional code Quest.

If mode is *pf/array* then the fix simply stores force values in an array. The fix adds these forces to each atom in the group, once every *Napply* steps, similar to the way the fix addforce command works.

The name of the public force array provided by the FixExternal class is

```
double **fexternal;
```

It is allocated by the FixExternal class as an (N,3) array where N is the number of atoms owned by a processor. The 3 corresponds to the fx, fy, fz components of force.

It is up to the external program to set the values in this array to the desired quantities, as often as desired. For example, the driver program might perform an MD run in stages of 1000 timesteps each. In between calls to the LIGGGHTS(R)-WSU run command, it could retrieve atom coordinates from LIGGGHTS(R)-WSU, compute forces, set values in fexternal, etc.

To use this fix during energy minimization, the energy corresponding to the added forces must also be set so as to be consistent with the added forces. Otherwise the minimization will not converge correctly.

This can be done from the external driver by calling this public method of the FixExternal class:

```
void set_energy(double eng);
```

where eng is the potential energy. Eng is an extensive quantity, meaning it should be the sum over per-atom energies of all affected atoms. It should also be provided in energy units consistent with the simulation. See the details below for how to insure this energy setting is used appropriately in a minimization.

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

No information about this fix is written to binary restart files.

The fix_modify *energy* option is supported by this fix to add the potential "energy" set by the external driver to the system's potential energy as part of thermodynamic output. This is a fictitious quantity but is needed so that the minimize command can include the forces added by this fix in a consistent manner. I.e. there is a decrease in potential energy when atoms move in the direction of the added force.

This fix computes a global scalar which can be accessed by various output commands. The scalar is the potential energy discussed above. The scalar stored by this fix is "extensive".

No parameter of this fix can be used with the *start/stop* keywords of the run command.

The forces due to this fix are imposed during an energy minimization, invoked by the minimize command.

IMPORTANT NOTE: If you want the fictitious potential energy associated with the added forces to be included in the total potential energy of the system (the quantity being minimized), you MUST enable the fix_modify *energy* option for this fix.

Restrictions: none

Related commands: none

Default: none

fix external command

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. The [granular pair styles](#) also detect if this fix has been defined and compute interactions between frozen and non-frozen particles appropriately, as if the frozen particle has infinite mass.

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

No information about this fix is written to [binary restart files](#). None of the [fix_modify](#) options are relevant to this fix.

This fix computes a global 3-vector of forces, which can be accessed by various [output commands](#). This is the total force on the group of atoms before the forces on individual atoms are changed by the fix. The vector values calculated by this fix are "extensive".

No parameter of this fix can be used with the *start/stop* keywords of the [run](#) command. This fix is not invoked during [energy minimization](#).

Restrictions:

There can only be a single freeze fix defined. This is because other the [granular pair styles](#) treat frozen particles differently and need to be able to reference a single group to which this fix is applied.

Related commands: none

[atom_style sphere](#)

Default: none

fix gravity command

Syntax:

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

- ID, group are documented in fix command
- gravity = style name of this fix command
- magnitude = size of acceleration (force/mass units)
- magnitude can be a variable (see below)
- style = *chute* or *spherical* or *gradient* or *vector*

```
  chute args = angle
      angle = angle in +x away from -z or -y axis in 3d/2d (in degrees)
      angle can be a variable (see below)
    spherical args = phi theta
      phi = azimuthal angle from +x axis (in degrees)
      theta = angle from +z or +y axis in 3d/2d (in degrees)
      phi or theta can be a variable (see below)
    vector args = x y z
      x y z = vector direction to apply the acceleration
      x or y or z can be a variable (see below)
```

Examples:

```
fix 1 all gravity 1.0 chute 24.0
fix 1 all gravity v_increase chute 24.0
fix 1 all gravity 1.0 spherical 0.0 -180.0
fix 1 all gravity 10.0 spherical v_phi v_theta
fix 1 all gravity 100.0 vector 1 1 0
```

Description:

Impose an additional acceleration on each particle in the group. This fix is typically used with granular systems to include a "gravity" term acting on the macroscopic particles. More generally, it can represent any kind of driving field, e.g. a pressure gradient inducing a Poiseuille flow in a fluid. Note that this fix operates differently than the fix addforce command. The addforce fix adds the same force to each atom, independent of its mass. This command imparts the same acceleration to each atom (force/mass).

The *magnitude* of the acceleration is specified in force/mass units. For granular systems (LJ units) this is typically 1.0. See the units command for details.

Style *chute* is typically used for simulations of chute flow where the specified *angle* is the chute angle, with flow occurring in the +x direction. For 3d systems, the tilt is away from the z axis; for 2d systems, the tilt is away from the y axis.

Style *spherical* allows an arbitrary 3d direction to be specified for the acceleration vector. *Phi* and *theta* are defined in the usual spherical coordinates. Thus for acceleration acting in the -z direction, *theta* would be 180.0 (or -180.0). *Theta* = 90.0 and *phi* = -90.0 would mean acceleration acts in the -y direction. For 2d systems, *phi* is ignored and *theta* is an angle in the xy plane where *theta* = 0.0 is the y-axis.

Style *vector* imposes an acceleration in the vector direction given by (x,y,z). Only the direction of the vector is important; it's length is ignored. For 2d systems, the *z* component is ignored.

Any of the quantities *magnitude*, *angle*, *phi*, *theta*, *x*, *y*, *z* which define the gravitational magnitude and direction, can be specified as an equal-style [variable](#). If the value is a variable, it should be specified as v_name, where name is the variable name. In this case, the variable will be evaluated each timestep, and its value used to determine the quantity. You should insure that the variable calculates a result in the approriate units, e.g. force/mass or degrees.

Equal-style variables can specify formulas with various mathematical functions, and include [thermo_style](#) command keywords for the simulation box parameters and timestep and elapsed time. Thus it is easy to specify a time-dependent gravitational field.

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

No information about this fix is written to [binary restart files](#).

The [fix_modify](#) *energy* option is supported by this fix to add the gravitational potential energy of the system to the system's potential energy as part of [thermodynamic output](#).

This fix computes a global scalar which can be accessed by various [output commands](#). This scalar is the gravitational potential energy of the particles in the defined field, namely mass * (g dot x) for each particles, where x and mass are the particles position and mass, and g is the gravitational field. The scalar value calculated by this fix is "extensive".

No parameter of this fix can be used with the *start/stop* keywords of the [run](#) command. This fix is not invoked during [energy minimization](#).

Restrictions: none

Related commands:

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

Default: none

fix heat/gran command

fix heat/gran/conduction command

Syntax:

```
fix ID group-ID heat/gran initial_temperature T0 keyword values
```

```
fix ID group-ID heat/gran/conduction initial_temperature T0 keyword values
```

- ID, group-ID are documented in fix command
- heat/gran/conduction or fix heat/gran = style name of this fix command
- initial_temperature = obligatory keyword
- T0 = initial (default) temperature for the particles
- zero or more keyword/value pairs may be appended
- keyword = *contact_area* or *area_correction*

  contact_area values = *overlap* or *constant* **value** or *projection*
    area_correction values = *yes* or *no*

Examples:

```
fix 3 hg heat/gran/conduction initial_temperature 273.15
```

LIGGGHTS(R)-WSU vs. LIGGGHTS(R)-WSU info:

This command is not available in LIGGGHTS(R)-WSU.

Description:

Calculates heat conduction between particles in contact and temperature update (see (Chaudhuri)) according to

$$\dot{Q}_{pi-pj} = h_{c,i-j}\, \Delta T_{pi-pj}$$

$$h_{c,i-j} = \frac{4 k_{pi} k_{pj}}{k_{pi} + k_{pj}} \left(A_{contact,i-j} \right)^{1/2}$$

$$m_p\, c_p\, \frac{dT_{p,i}}{dt} = \underbrace{\sum_{contacts\ i-j} \dot{Q}_{pi-pj}}_{\substack{heat\ conduction \\ by\ contacts}} + \underbrace{\dot{Q}_{pi,source}}_{\substack{heat\ generation \\ due\ to\ sources, \\ e.g.\ reactions}}$$

h_c heat transfer coefficient $\left[\frac{J}{K s} \right]$

k_{pi} thermal conductivity of particle i $\left[\frac{J}{K s m} \right]$

c_p specific thermal capacity $\left[\frac{J}{kg K} \right]$

$A_{contact,i-j}$ particle contact area $[m^2]$

fix heat/gran command   212

It is assumed that the temperature within the particles is uniform. To make particles adiabatic (so they do not change the temperature), do not include them in the fix group. However, heat transfer is calculated between particles in the group and particles not in the group (but temperature update is not performed for particles not in the group). Thermal conductivity and specific thermal capacity must be defined for each atom type used in the simulation by means of fix property/global commands:

```
fix id all property/global thermalConductivity peratomtype value_1 value_2 ...
(value_i=value for thermal conductivity of atom type i)

fix id all property/global thermalCapacity peratomtype value_1 value_2 ...
(value_i=value for thermal capacity of atom type i)
```

To set the temperature for a group of particles, you can use the set command with keyword *property/atom* and values *Temp T*. *T* is the temperature value you want the particles to have. To set heat sources (or sinks) for a group of particles, you can also use the set command with the set keyword: *property/atom* and the set values: *heatSource h* where *h* is the heat source value you want the particles to have (in Energy/time units). A negative value means it is a heat sink. Examples would be:

```
set region halfbed property/peratom Temp 800.
set region srcreg property/peratom heatSource 0.5
```

Contact area calculation:

Using keyword *contact_area*, you can choose from 3 modes of calulating the contact area for particle-particle heat transfer: If *overlap* is used, the contact area is calculated from the area of the sphere-sphere intersection If *constant* is used, a constant user-defined overlap area is assumed. If *projection* is used, the overlap area is assumed to be equal to rmin*rmin*Pi, where rmin is the radius of the smaller of the two particles in contact.

Area correction:

For *contact_area* = overlap, an area correction can additionally be performed using keyword *area_correction* to account for the fact that the Young's modulus might have been decreased in order to speed-up the simulation, thus artificially increasing the overlap. In this case, you have to specify the original Young's modulus of each material by means of a fix property/global command:

```
fix id all property/global youngsModulusOriginal peratomtype value_1 value_2 ...
(value_i=value for original Young's modulus of atom type i)
```

This area correction is performed by scaling the contact area with $(Y^*/Y^*,orig)^a$, where Y^* and $Y^*,orig$ are calculated as defined in pair_style gran . The scaling factor is given as e.g. a=1 for a Hooke and a=2/3 for a Hertz interaction.

Output info:

You can visualize the heat sources by accessing f_heatSource[0], and the heatFluxes by f_heatFlux[0] . With f_directionalHeatFlux[0], f_directionalHeatFlux[1] and f_directionalHeatFlux[2] you can access the conductive heat fluxes in x,y,z directions. The conductive heat fluxes are calculated per-contact and half the value is stored in each atom participating in the contact. With f_Temp[0] you can access the per-particle temperature. You can also access the total thermal energy of the fix group (useful for the thermo command) with f_id .

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

The particle temperature and heat source is written is written to binary restart files so simulations can continue properly. None of the fix_modify options are relevant to this fix.

fix heat/gran/conduction command

This fix computes a scalar which can be accessed by various output commands. This scalar is the total thermal energy of the fix group

No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions:

Related commands:

compute temp, compute temp/region

Default:

contact_area = overlap, *area_correction* = no

Literature:

(Chaudhuri) Chaudhuri et al, Chemical Engineering Science, 61, p 6348 (2006).

fix command

Syntax:

```
fix ID group-ID style args
```

- ID = user-assigned name for the fix
- group-ID = ID of the group of atoms to apply the fix to
- style = one of a long list of possible style names (see below)
- args = arguments used by a particular style

Examples:

```
fix 1 all nve
fix 3 all nvt temp 300.0 300.0 0.01
fix mine top setforce 0.0 NULL 0.0
```

Description:

Set a fix that will be applied to a group of atoms. In LIGGGHTS(R)-WSU, 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 LIGGGHTS(R)-WSU and new ones can be added; see [this section](#) for a discussion.

The full list of fixes defined in LIGGGHTS(R)-WSU is on [this page](#).

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

The ID of a fix can only contain alphanumeric characters and underscores.

Fixes can be deleted with the [unfix](#) command.

IMPORTANT NOTE: The [unfix](#) command is the only way to turn off a fix; simply specifying a new fix with a similar style will not turn off the first one. This is especially important to realize for integration fixes. For example, using a [fix nve](#) command for a second run after using a [fix nvt](#) command for the first run, will not cancel out the NVT time integration invoked by the "fix nvt" command. Thus two time integrators would be in place!

If you specify a new fix with the same ID and style as an existing fix, the old fix is deleted and the new one is created (presumably with new settings). This is the same as if an "unfix" command were first performed on the old fix, except that the new fix is kept in the same order relative to the existing fixes as the old one originally was. Note that this operation also wipes out any additional changes made to the old fix via the [fix_modify](#) command.

The [fix modify](#) command allows settings for some fixes to be reset. See the doc page for individual fixes for details.

Some fixes store an internal "state" which is written to binary restart files via the [restart](#) or [write_restart](#) commands. This allows the fix to continue on with its calculations in a restarted simulation. See the [read_restart](#) command for info on how to re-specify a fix in an input script that reads a restart file. See the doc

pages for individual fixes for info on which ones can be restarted.

Some fixes calculate one of three styles of quantities: global, per-atom, or local, which can be used by other commands or output as described below. A global quantity is one or more system-wide values, e.g. the energy of a wall interacting with particles. A per-atom quantity is one or more values per atom, e.g. the displacement vector for each atom since time 0. Per-atom values are set to 0.0 for atoms not in the specified fix group. Local quantities are calculated by each processor based on the atoms it owns, but there may be zero or more per atoms.

Note that a single fix may produces either global or per-atom or local quantities (or none at all), but never more than one of these.

Global, per-atom, and local quantities each come in three kinds: a single scalar value, a vector of values, or a 2d array of values. The doc page for each fix describes the style and kind of values it produces, e.g. a per-atom vector. Some fixes produce more than one kind of a single style, e.g. a global scalar and a global vector.

When a fix quantity is accessed, as in many of the output commands discussed below, it can be referenced via the following bracket notation, where ID is the ID of the fix:

f_ID	entire scalar, vector, or array
f_ID[I]	one element of vector, one column of array
f_ID[I][J]	one element of array

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

Note that commands and variables which use fix quantities typically do not allow for all kinds, e.g. a command may require a vector of values, not a scalar. This means there is no ambiguity about referring to a fix quantity as f_ID even if it produces, for example, both a scalar and vector. The doc pages for various commands explain the details.

In LIGGGHTS(R)-WSU, the values generated by a fix can be used in several ways:

- Global values can be output via the thermo_style custom or fix ave/time command. Or the values can be referenced in a variable equal or variable atom command.
- Per-atom values can be output via the dump custom command or the fix ave/spatial command. Or they can be time-averaged via the fix ave/atom command or reduced by the compute reduce command. Or the per-atom values can be referenced in an atom-style variable.
- Local values can be reduced by the compute reduce command, or histogrammed by the fix ave/histo command.

See this howto section for a summary of various LIGGGHTS(R)-WSU output options, many of which involve fixes.

The results of fixes that calculate global quantities can be either "intensive" or "extensive" values. Intensive means the value is independent of the number of atoms in the simulation, e.g. temperature. Extensive means the value scales with the number of atoms in the simulation, e.g. total rotational kinetic energy. Thermodynamic output will normalize extensive values by the number of atoms in the system, depending on the "thermo_modify norm" setting. It will not normalize intensive values. If a fix value is accessed in another way, e.g. by a variable, you may want to know whether it is an intensive or extensive value. See the doc page for individual fixes for further info.

Each fix style has its own documentation page which describes its arguments and what it does, as listed below.

The full list of fixes defined in LIGGGHTS(R)-WSU is on this page.

Restrictions:

Some fix styles are part of specific packages. They are only enabled if LIGGGHTS(R)-WSU was built with that package. See the Making LIGGGHTS(R)-WSU section for more info on packages. The doc pages for individual fixes tell if it is part of a package.

Related commands:

unfix, fix_modify

Default: none

fix insert/pack command

Syntax:

```
fix ID group-ID insert/pack seed seed_value distributiontemplate dist-ID general_keywords general
```

- ID, group-ID are documented in fix command
- insert/pack = style name of this fix command
- seed = obligatory keyword
- seed_value = random # seed (positive integer)
- distributiontemplate = obligatory keyword
- dist-ID = ID of a fix_particledistribution_discrete to be used for particle insertion
- one or more general keyword/value pairs can be appended
- general_keywords = *verbose* or *maxattempt* or *insert_every* or *overlapcheck* or *all_in* or *random_distribute* or *vel constant* or *vel uniform* or *vel gaussian* or *orientation* or *omega* or *set_property*

```
verbose = yes or no
  maxattempt value = ma
    ma = max # of insertion attempts per atom (positive integer)
  insert_every value = once or ie
    once = value to signalise that isertion takes place only once (the step after the fix
    ie = every how many time-steps particles are inserted - insertion happens periodically
  start value = ts
    ts = time-step at which insertion should start (positive integer larger than current t
  overlapcheck value = yes or no
  all_in value = yes or no
  random_distribute value = exact or uncorrelated
  vel constant values = vx vy vz
    vx = x-velocity at insertion (velocity units)
    vy = y-velocity at insertion (velocity units)
    vz = z-velocity at insertion (velocity units)
  vel uniform values = vx vy vz vFluctx vFlucty vFluctz
    vx = mean x-velocity at insertion (velocity units)
    vy = mean y-velocity at insertion (velocity units)
    vz = mean z-velocity at insertion (velocity units)
    vFluctx = amplitude of uniform x-velocity fluctuation at insertion (velocity units)
    vFlucty = amplitude of uniform y-velocity fluctuation at insertion (velocity units)
    vFluctz = amplitude of uniform z-velocity fluctuation at insertion (velocity units)
  vel gaussian values = vx vy vz vFluctx vFlucty vFluctz
    vx = mean x-velocity at insertion (velocity units)
    vy = mean y-velocity at insertion (velocity units)
    vz = mean z-velocity at insertion (velocity units)
    vFluctx = standard deviation of Gaussian x-velocity fluctuation at insertion (velocity
    vFlucty = standard deviation of Gaussian y-velocity fluctuation at insertion (velocity
    vFluctz = standard deviation of Gaussian z-velocity fluctuation at insertion (velocity
  orientation values = random or template
    random = randomize rotational orientation
    template = use orientation from particle template
  omega values = constant omegax omegay omegaz
    constant = obligatory word
    omegax = x-comonent of angular velocity (1/time units)
    omegay = y-comonent of angular velocity (1/time units)
    omegaz = z-comonent of angular velocity (1/time units)
  set_property values = property-ID val
    property-ID = ID of a fix property/atom holding a scalar value for each particle
    val = value to initialize the property with upon insertion
```

- following the general keyword/value section, one or more pack keyword/value pairs can be appended for the fix insert/pack command

- pack_keywords = *region* or *volumefraction_region* or *particles_in_region* or *mass_in_region* or *ntry_mc*

```
pack_keywords = where exactly one out of volumefraction_region or particles_in_region or ma
  region value = region-ID
    region-ID = ID of the region where the particles will be generated
  volumefraction_region values = vol
    vol =  desired volume fraction for the region (positive float, 0 <vol  <1)
  particles_in_region values = np
    np =  desired number of particles in the region (positive integer)
  mass_in_region values = m
    m =  desired mass in the region (positive float, m > 0)
  ntry_mc values = n
    n = number of Monte-Carlo steps for calculating the region's volume  (positive integer
```

Examples:

```
fix ins all insert/pack seed 1001 distributiontemplate pdd1 insert_every once overlapcheck yes vo
```

Description:

Insert particles into a granular run either once or every few timesteps within the specified region, as defined via the *region* keyword.

The *verbose* keyword controlls whether statistics about particle insertion is output to the screen each time particles are inserted.

This command must use the distributiontemplate keyword to refer to a fix_particledistribution_discrete (defined by dist-fix-ID) that defines the properties of the inserted particles.

At each insertion step, fix insert/pack tries inserts as many particles as needed to reach a defined target, which can be either a region volume fraction (keyword *volumefraction_region*), the total number of particles in the region (keyword *particles_in_region*), or the total particle mass in the region (keyword *mass_in_region*). Exactly one out of the keywords *volumefraction_region*, *particles_in_region*, *mass_in_region* must be defined.

The frequency of the particle insertion can be controlled by the keyword *insert_every*, which defines the number of time-steps between two insertions. Alternatively, by specifying *insert_every once*, particles are inserted only once.

The *start* keyword can be used to set the time-step at which the insertion should start.

Inserted particles are assigned the atom type specified by the particledistribution defined via the fix_particledistribution_discrete and are assigned to 4 groups: the default group "all" and the group specified in the fix insert command, as well as the groups specified in the fix_particledistribution_discrete and fix_particletemplate_sphere command (all of which can also be "all").

The keyword *overlapcheck* controls if overlap is checked for at insertion, both within the inserted particle package and with other existig particles. If this option is turned off, insertion will scale very well in parallel, otherwise not. Be aware that in case of no overlap check, highly overlapping configurations will be produced, so you will have to relax these configurations.

If overlapcheck if performed, the number of insertion attempts per particle can be specified via the *maxattempt* keyword. Each timestep particles are inserted, the command will make up to a total of M tries to insert the new particles without overlaps, where M = # of inserted particles * ma. If unsuccessful at completing all insertions, a warning will be printed.

fix insert/pack command

The *all_in* flag determines if the particle is completely contained in the insertion region (*all_in yes*) or only the particle center (*all_in no*). Currently *all_in yes* is not yet supported for all types of insertion.

Keyword *random_distribute* controls how the number of particles to be inserted is distributed among parallel processors and among the particle templates in the particle distribution. For style *exact*, the number of particles to be inserted each step is matched exactly. For style *uncorrelated*, the number of particles to be inserted for each particle template will be rounded in an uncorrelated way, so the total number of inserted particles may vary for each insertion step. However, statistically both ways should produce the same result. Style *uncorrelated* may be faster in parallel since it does not need global MPI operations. Please note that if the # of particles to be inserted is calculated e.g. from a particle mass to be inserted, the number of particles to be inserted each insertion step will vary by 1, irrespective of the *random_distribute* settings. This is because in this case the # of particles to insert in each step will be a floating point number, and applying a simple floor/ceil rounding operation would lead to a statistical bias.

The initial velocity and rotational velocity can be controlled via the *vel* and *omega* keywords. *vel constant* simply patches a constant velocity to the inserted particles, *vel uniform* sets uniformly distributed velocities with mean and amplitude. *vel gaussian* sets Gaussian distributed particle velocities with mean and std. deviation.

The *set_property* option can be used to initialize scalar per-particle properties such as temperatures, which are stored in a a fix property/atom.

Description for fix insert/pack:

This command must use the *region* keyword to define an insertion volume. The specified region must have been previously defined with a region command. Dynamic regions are not supported as insertion region. Each timestep particles are inserted, they are placed randomly inside the insertion volume.

The *volumefraction* 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 should not overlap, the maximum volume fraction should be no higher than about 0.6.

To determine the volume of the insertion region, a Monte Carlo approach might be used for some cases where the volume is difficult to calculate or where the volume calculation is simply not implemented by the region. The *ntry_mc* keyword is used to control the number of MC tries that are used for the volume calculation.

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

Information about this fix is written to binary restart files. This means you can restart a simulation while inserting particles, when the restart file was written during the insertion operation.

None of the fix_modify options are relevant to this fix. A global vector is stored by this fix for access by various output commands. The first component of the vector is the number of particles already inserted, the second component is the mass of particles already inserted. No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions:

The *overlapcheck* = 'yes' option performs an inherently serial operation and will thus not scale well in parallel. For this reason, if you want to generate large systems, you are advised to turn *overlapcheck* off and let the packing relax afterwards to generate a valid packing.

Option *all_in* = 'yes' will not work if the region used is a tet mesh region.

Keywords *duration* and *extrude_length* can not be used together.

Currently *all_in yes* is not yet supported for all types of insertion.

Dynamic regions are not supported as insertion region.

Related commands:

fix_insert_stream, fix_insert_rate_region, region

Default:

The defaults are maxattempt = 50, all_in = no, overlapcheck = yes vel = 0.0 0.0 0.0, omega = 0.0 0.0 0.0, start = next time-step, duration = insert_every, ntry_mc = 100000, random_distribute = exact

fix insert/rate/region command

Syntax:

```
fix ID group-ID insert/rate/region   seed seed_value distributiontemplate dist-ID general_keyword
```

- ID, group-ID are documented in [fix](#) command
- insert/pack and insert/stream = style names of these fix commands
- seed = obligatory keyword
- seed_value = random # seed (positive integer)
- distributiontemplate = obligatory keyword
- dist-ID = ID of a [fix_particledistribution_discrete](#) to be used for particle insertion
- one or more general keyword/value pairs can be appended
- general_keywords = *verbose* or *maxattampt* or *nparticles* or *mass* or *particlerate* or *massrate* or *insert_every* or *overlapcheck* or *all_in* or *random_distribute* or *vel constant* or *vel uniform* or *vel gaussian* or *orientation* or *omega* or *set_property*

```
verbose = yes or no
  maxattempt value = ma
    ma = max # of insertion attempts per atom (positive integer)
  nparticles values = np or INF
    np =  number of particles to insert (positive integer)
    INF =  insert as many particles as possible
  mass values = mp
    mp = mass of particles to be inserted (positive float)
    INF =  insert as many particles as possible
  particlerate values = pr
    pr = particle inseration rate (particles/time units)
  massrate values = mr
    mr = mass inseration rate (mass/time units)
  insert_every value = once or ie
    once = value to signalise that isertion takes place only once (the step after the fix
    ie = every how many time-steps particles are inserted - insertion happens periodically
  start value = ts
    ts = time-step at which insertion should start (positive integer larger than current ti
  overlapcheck value = yes or no
  all_in value = yes or no
  random_distribute value = exact or uncorrelated
  vel constant values = vx vy vz
    vx = x-velocity at insertion (velocity units)
    vy = y-velocity at insertion (velocity units)
    vz = z-velocity at insertion (velocity units)
  vel uniform values = vx vy vz vFluctx vFlucty vFluctz
    vx = mean x-velocity at insertion (velocity units)
    vy = mean y-velocity at insertion (velocity units)
    vz = mean z-velocity at insertion (velocity units)
    vFluctx = amplitude of uniform x-velocity fluctuation at insertion (velocity units)
    vFlucty = amplitude of uniform y-velocity fluctuation at insertion (velocity units)
    vFluctz = amplitude of uniform z-velocity fluctuation at insertion (velocity units)
  vel gaussian values = vx vy vz vFluctx vFlucty vFluctz
    vx = mean x-velocity at insertion (velocity units)
    vy = mean y-velocity at insertion (velocity units)
    vz = mean z-velocity at insertion (velocity units)
    vFluctx = standard deviation of Gaussian x-velocity fluctuation at insertion (velocity
    vFlucty = standard deviation of Gaussian y-velocity fluctuation at insertion (velocity
    vFluctz = standard deviation of Gaussian z-velocity fluctuation at insertion (velocity
  orientation values = random or template
    random = randomize rotational orientation
    template = use orientation from particle template
```

```
    omega values = constant omegax omegay omegaz
      constant = obligatory word
      omegax = x-comonent of angular velocity (1/time units)
      omegay = y-comonent of angular velocity (1/time units)
      omegaz = z-comonent of angular velocity (1/time units)
    set_property values = property-ID val
      property-ID = ID of a fix property/atom holding a scalar value for each particle
      val = value to initialize the property with upon insertion
```

- following the general keyword/value section, one or more rate_region keyword/value pairs can be appended for the fix insert/rate/region command
- rate_region keywords = *region* or *ntry_mc*

```
  region value = region-ID
      region-ID = ID of the region where the particles will be generated (positive integer)
    ntry_mc values = n
      n = number of Monte-Carlo steps for calculating the region's volume  (positive integer
```

Examples:

```
fix ins all insert/rate/region seed 1001 distributiontemplate pdd1 nparticles 1000 particlerate 5
```

General description:

Insert particles into a granular run every few timesteps within a specified region at a specified rate.

The *verbose* keyword controlls whether statistics about particle insertion is output to the screen each time particles are inserted.

This command must use the *region* keyword to define an insertion volume. The specified region must have been previously defined with a region command. Dynamic regions are not supported as insertion region. Each timestep particles are inserted, they are placed randomly inside the insertion volume.

To specify the number of particles to be inserted, you must use either the *nparticles* or the *mass* keyword (but not both). In the latter case, the number of particles to be inserted is calculated from the mass expectancy given by the particle distribution.

Likewise, you can use the *particlerate* or the *massrate* keyword (but not both) to control the insertion rate.

The frequency of the particle insertion is controlled by the keyword *insert_every*, which defines the number of time-steps between two insertions. The number of particles to be inserted at each insertion event is calculated from the insertion rate and *insert_every*. The *start* keyword can be used to set the time-step at which the insertion should start.

This command must use the distributiontemplate keyword to refer to a fix_particledistribution_discrete (defined by dist-fix-ID) that defines the properties of the inserted particles.

Inserted particles are assigned the atom type specified by the particledistribution defined via the fix_particledistribution_discrete and are assigned to 4 groups: the default group "all" and the group specified in the fix insert command, as well as the groups specified in the fix_particledistribution_discrete and fix_particletemplate_sphere command (all of which can also be "all").

The keyword *overlapcheck* controls if overlap is checked for at insertion, both within the inserted particle package and with other existig particles. If this option is turned off, insertion will scale very well in parallel, otherwise not. Be aware that in case of no overlap check, highly overlapping configurations will be produced, so you will have to relax these configurations.

If overlapcheck if performed, the number of insertion attempts per particle can be specified via the *maxattempt* keyword. Each timestep particles are inserted, the command will make up to a total of M tries to insert the new particles without overlaps, where M = # of inserted particles * ma. If unsuccessful at completing all insertions, a warning will be printed.

The *all_in* flag determines if the particle is completely contained in the insertion region (*all_in yes*) or only the particle center (*all_in no*). Currently *all_in yes* is not yet supported for all types of insertion.

Keyword *random_distribute* controls how the number of particles to be inserted is distributed among parallel processors and among the particle templates in the particle distribution. For style *exact*, the number of particles to be inserted each step is matched exactly. For style *uncorrelated*, the number of particles to be inserted for each particle template will be rounded in an uncorrelated way, so the total number of inserted particles may vary for each insertion step. However, statistically both ways should produce the same result. Style *uncorrelated* may be faster in parallel since it does not need global MPI operations. Please note that if the # of particles to be inserted is calculated e.g. from a particle mass to be inserted, the number of particles to be inserted each insertion step will vary by 1, irrespective of the *random_distribute* settings. This is because in this case the # of particles to insert in each step will be a floating point number, and applying a simple floor/ceil rounding operation would lead to a statistical bias.

The initial velocity and rotational velocity can be controlled via the *vel* and *omega* keywords. *vel constant* simply patches a constant velocity to the inserted particles, *vel uniform* sets uniformly distributed velocities with mean and amplitude. *vel gaussian* sets Gaussian distributed particle velocities with mean and std. deviation.

The *set_property* option can be used to initialize scalar per-particle properties such as temperatures, which are stored in a a [fix property/atom](#).

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

Information about this fix is written to [binary restart files](#). This means you can restart a simulation while inserting particles, when the restart file was written during the insertion operation.

None of the [fix_modify](#) options are relevant to this fix. A global vector is stored by this fix for access by various [output commands](#). The first component of the vector is the number of particles already inserted, the second component is the mass of particles already inserted. No parameter of this fix can be used with the *start/stop* keywords of the [run](#) command. This fix is not invoked during [energy minimization](#).

Restrictions:

The *overlapcheck* = 'yes' option performs an inherently serial operation and will thus not scale well in parallel. For this reason, if you want to generate large systems, you are advised to turn *overlapcheck* off and let the packing relax afterwards to generate a valid packing.

Option *all_in* = 'yes' will not work if the [region](#) used is a tet mesh region.

Keywords *duration* and *extrude_length* can not be used together.

Currently *all_in yes* is not yet supported for all types of insertion.

Dynamic regions are not supported as insertion region.

Related commands:

[fix_insert_stream](#), [fix_insert_pack](#), [fix_gravity](#), [region](#)

Default:

The defaults are maxattempt = 50, all_in = no, overlapcheck = yes vel = 0.0 0.0 0.0, omega = 0.0 0.0 0.0, start = next time-step, duration = insert_every, ntry_mc = 100000, random_distribute = exact

fix insert/stream command

Syntax:

```
fix ID group-ID insert/stream seed seed_value distributiontemplate dist-ID general_keywords gener
```

- ID, group-ID are documented in <u>fix</u> command
- insert/stream = style name of this fix command
- seed = obligatory keyword
- seed_value = random # seed (positive integer)
- distributiontemplate = obligatory keyword
- dist-ID = ID of a <u>fix_particledistribution_discrete</u> to be used for particle insertion
- one or more general keyword/value pairs can be appended
- general_keywords = *verbose* or *maxattampt* or *nparticles* or *mass* or *particlerate* or *massrate* or *insert_every* or *overlapcheck* or *all_in* or *random_distribute* or *vel constant* or *vel uniform* or *vel gaussian* or *orientation* or *omega* or *set_property*

```
verbose = yes or no
  maxattempt value = ma
    ma = max # of insertion attempts per atom (positive integer)
  nparticles values = np or INF
    np =  number of particles to insert (positive integer)
    INF =  insert as many particles as possible
  mass values = mp
    mp = mass of particles to be inserted (positive float)
    INF =  insert as many particles as possible
  particlerate values = pr
    pr = particle inseration rate (particles/time units)
  massrate values = mr
    mr = mass inseration rate (mass/time units)
  insert_every value = ie
    ie = every how many time-steps particles are inserted – insertion happens periodically
  start value = ts
    ts = time-step at which insertion should start (positive integer larger than current t
  overlapcheck value = yes or no
  all_in value = yes or no
  random_distribute value = exact or uncorrelated
  vel constant values = vx vy vz
    vx = x-velocity at insertion (velocity units)
    vy = y-velocity at insertion (velocity units)
    vz = z-velocity at insertion (velocity units)
  vel uniform values = vx vy vz vFluctx vFlucty vFluctz
    vx = mean x-velocity at insertion (velocity units)
    vy = mean y-velocity at insertion (velocity units)
    vz = mean z-velocity at insertion (velocity units)
    vFluctx = amplitude of uniform x-velocity fluctuation at insertion (velocity units)
    vFlucty = amplitude of uniform y-velocity fluctuation at insertion (velocity units)
    vFluctz = amplitude of uniform z-velocity fluctuation at insertion (velocity units)
  vel gaussian values = vx vy vz vFluctx vFlucty vFluctz
    vx = mean x-velocity at insertion (velocity units)
    vy = mean y-velocity at insertion (velocity units)
    vz = mean z-velocity at insertion (velocity units)
    vFluctx = standard deviation of Gaussian x-velocity fluctuation at insertion (velocity
    vFlucty = standard deviation of Gaussian y-velocity fluctuation at insertion (velocity
    vFluctz = standard deviation of Gaussian z-velocity fluctuation at insertion (velocity
  orientation values = random or template
    random = randomize rotational orientation
    template = use orientation from particle template
  omega values = constant omegax omegay omegaz
```

```
        constant = obligatory word
        omegax = x-comonent of angular velocity (1/time units)
        omegay = y-comonent of angular velocity (1/time units)
        omegaz = z-comonent of angular velocity (1/time units)
    set_property values = property-ID val
        property-ID = ID of a fix property/atom holding a scalar value for each particle
        val = value to initialize the property with upon insertion
```

- following the general keyword/value section, one or more stream keyword/value pairs can be appended for the fix insert/stream command
- stream_keywords = *duration* or *parallel* or *insertion_face* or *extrude_length*

```
    insertion_face value = mesh-ID
        mesh-ID = ID of the fix mesh/surface or fix mesh/surface/planar to use as starting face
    extrude_length values = L
        L =  length for extruding the insertion face in normal direction so to generate in inse
    parallel values = yes or no
        yes, no = pre-calculate location of overlap of processor subdomains and extrusion volu
    duration values = du
        du =  duration of insertion in time-steps
```

Examples:

fix ins all insert/stream seed 1001 distributiontemplate pdd1 nparticles 5000 vel constant 0. -0.5 -2. particlerate 1000 overlapcheck yes insertion_face ins_mesh extrude_length 0.6

Description:

Insert particles into a granular run either once or every few timesteps within a specified region until either np particles have been inserted or the desired particle mass (mp) has been reached.

The *verbose* keyword controlls whether statistics about particle insertion is output to the screen each time particles are inserted.

Each timestep particles are inserted, they are placed randomly inside the insertion volume so as to mimic a stream of poured particles. The insertion volume is generated by extruding the insertion face as specified via *insertion_face* in the direction of the face normal. The sign of this face normal is automatically flipped so that it is opposite to the normal component of the insertion velocity.

To specify the number of particles to be inserted, you must use either the *nparticles* or the *mass* keyword (but not both). In the latter case, the number of particles to be inserted is calculated from the mass expectancy given by the particle distribution. The *start* keyword can be used to set the time-step at which the insertion should start.

Likewise, you can use the *particlerate* or the *massrate* keyword (but not both) to control the insertion rate. Particles are not inserted continuously, but in packets (for efficiency reasons). Particles are inserted again after enough time has elapsed that the previously inserted particles have left the insertion volume.

One of the two keywords *insert_every* and *extrude_length* must be provided by the user (but not both).

In case *insert_every* is defined, this sets the frequency of the particle insertion directly, i.e. the number of time-steps between two insertions. The number of particles to be inserted at each insertion event is calculated from the insertion rate and *insert_every*.

If *extrude_length* is specified, the amount of extrusion is fixed and the insertion frequency is calculated from *extrude_length* and the insertion velocity normal to the insertion face.

When defining *insert_every*, you have the possibility to define the duration of each insertion via the *duration* keyword. *duration < insert_every* will generate a "pulsed" stream as opposed to a continuous stream. Example: Setting *insert_every* = 1000 and *duration* = 600 will produce a stream that pours material for 600 time-steps, will pause for 400 time-steps, pour for another 600 time-steps etc.

As mentioned above, particles are inserted again after enough time has elapsed that the previously inserted particles have left the insertion volume. Until the time these particles reach the insertion face, no other forces affect the particles (pair forces, gravity etc.). Fix insert/stream internally issues a special integrator to take care of this. This procedure guarantees that the specified velocity, omega etc. values are perfectly met at the specified insertion face.

The larger the volume, the more particles that can be inserted at one insertion step. Insertions will continue until the desired # of particles has been inserted.

NOTE: The insertion face must be a planar face, and the insertion velocity projected on the face normal must be non-zero.

NOTE: Keywords *insert_every* and *extrude_length* may not be used together

NOTE: Keywords *duration* and *extrude_length* cannot be used together.

This command must use the distributiontemplate keyword to refer to a fix_particledistribution_discrete (defined by dist-fix-ID) that defines the properties of the inserted particles.

Inserted particles are assigned the atom type specified by the particledistribution defined via the fix_particledistribution_discrete and are assigned to 4 groups: the default group "all" and the group specified in the fix insert command, as well as the groups specified in the fix_particledistribution_discrete and fix_particletemplate_sphere command (all of which can also be "all").

The keyword *overlapcheck* controls if overlap is checked for at insertion, both within the inserted particle package and with other existig particles. If this option is turned off, insertion will scale very well in parallel, otherwise not. Be aware that in case of no overlap check, highly overlapping configurations will be produced, so you will have to relax these configurations.

If overlapcheck if performed, the number of insertion attempts per particle can be specified via the *maxattempt* keyword. Each timestep particles are inserted, the command will make up to a total of M tries to insert the new particles without overlaps, where M = # of inserted particles * ma. If unsuccessful at completing all insertions, a warning will be printed.

The *all_in* flag determines if the particle is completely contained in the insertion region (*all_in* = *yes*) or only the particle center (*all_in* = *no*).Using *all_in* = yes requires you to use an insertion face of style [fix mesh/surface/planar](fix_mesh/surface/planar)

NOTE: You also have to use [fix mesh/surface/planar](fix_mesh/surface/planar) if there is a [run](run) command between the definition of the insertion face and the fix insert/stream command. Otherwise, a [fix mesh/surface/planar](fix_mesh/surface/planar) will do.

Keyword *random_distribute* controls how the number of particles to be inserted is distributed among parallel processors and among the particle templates in the particle distribution. For style *exact*, the number of particles to be inserted each step is matched exactly. For style *uncorrelated*, the number of particles to be inserted for each particle template will be rounded in an uncorrelated way, so the total number of inserted particles may vary for each insertion step. However, statistically both ways should produce the same result. Style *uncorrelated* may be faster in parallel since it does not need global MPI operations. Please note that if the # of particles to be inserted is calculated e.g. from a particle mass to be inserted, the number of particles to be inserted each insertion step will vary by 1, irrespective of the *random_distribute* settings. This is because in

this case the # of particles to insert in each step will be a floating point number, and applying a simple floor/ceil rounding operation would lead to a statistical bias.

If keyword *parallel* is set to 'yes', LIGGGHTS(R)-WSU tries to pre-calculate more accurately the overlap of process subdomains and extrusion volume. For cases where the insertion volume is highly divided between different processes, this can lead to a speed-up of insertion as random number generation is more efficient. For cases where the extrusion volume is divided among few processes this will impose a small computation overhead.

The initial velocity and rotational velocity can be controlled via the *vel* and *omega* keywords. *vel constant* simply patches a constant velocity to the inserted particles, *vel uniform* sets uniformly distributed velocities with mean and amplitude. *vel gaussian* sets Gaussian distributed particle velocities with mean and std. deviation. The insertion velocity must be non-zero.

The *set_property* option can be used to initialize scalar per-particle properties such as temperatures, which are stored in a a [fix property/atom](#).

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

Information about this fix is written to [binary restart files](#). This means you can restart a simulation simulation while inserting particles, when the restart file was written during the insertion operation.

None of the [fix_modify](#) options are relevant to this fix. A global vector is stored by this fix for access by various [output commands](#). The first component of the vector is the number of particles already inserted, the second component is the mass of particles already inserted. No parameter of this fix can be used with the *start/stop* keywords of the [run](#) command. This fix is not invoked during [energy minimization](#).

Restrictions:

Keywords *duration* and *extrude_length* can not be used together. The insertion face cannot move.

Related commands:

[fix_insert_pack](#), [fix_insert_rate_region](#),

Default:

The defaults are maxattempt = 50, all_in = no, overlapcheck = yes vel = 0.0 0.0 0.0, omega = 0.0 0.0 0.0, start = next time-step, duration = insert_every, random_distribute = exact, parallel = no

fix lineforce command

Syntax:

```
fix ID group-ID lineforce x y z
```

- ID, group-ID are documented in [fix](#) command
- lineforce = style name of this fix command
- x y z = direction of line as a 3-vector

Examples:

```
fix hold boundary lineforce 0.0 1.0 1.0
```

Description:

Adjust the forces on each atom in the group so that only the component of force along the linear direction specified by the vector (x,y,z) remains. This is done by subtracting out components of force in the plane perpendicular to the line.

If the initial velocity of the atom is 0.0 (or along the line), then it should continue to move along the line thereafter.

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

No information about this fix is written to [binary restart files](#). None of the [fix_modify](#) options are relevant to this fix. No global or per-atom quantities are stored by this fix for access by various [output commands](#). No parameter of this fix can be used with the *start/stop* keywords of the [run](#) command.

The forces due to this fix are imposed during an energy minimization, invoked by the [minimize](#) command.

Restrictions: none

Related commands:

[fix planeforce](#)

Default: none

fix massflow/mesh command

Syntax:

```
fix id group massflow/mesh mesh mesh-ID vec_side vx vy vz keyword value ...
```

- ID, group-ID are documented in fix command
- massflow/mesh = style name of this fix command
- mesh = obligatory keyword
- mesh-ID = ID of a fix mesh/surface command
- vec_side = obligatory keyword
- vx, vy, vz = vector components defining the "outside" of the mesh
- zero or more keyword/value pairs may be appended to args
- keywords = *count* or *point_at_outlet* or *append* or *file* or *screen* or *delete_atoms*

```
  count value = once or multiple
     once = count particles only once
     multiple = allow particles to be counted multiple times
  point_at_outlet  pointX pointY pointZ
    pointX pointY pointZ = coordinates of point on the outlet side of the surface
  inside_out
    use this in connection with point_at_outlet to flip direction particle counting
  file value = filename
  append value = filename
   filename = name of the file to print diameter, position and velocity values of the part
  screen value = yes or no
  writeTime
    include this keyword to write the time to the out files
  delete_atoms value = yes or no
    yes = to remove the particles that pass through the mesh surface
```

Examples:

```
fix mass all massflow/mesh mesh inface vec_side 0. 0. -1.

fix mass all massflow/mesh mesh inface count once point_at_outlet 0. 0. 0.
```

Description:

Fix massflow/mesh tracks how many particles penetrate through a mesh surface, as defined by a fix mesh/surface command. It counts the total number of particles and the associated mass. Only particles part of *group* are eligible for counting.

Particles are counted if they cross from the inner side of the mesh to the outer side of the mesh. The outer side can be defined by using the keyword *vec_side*, by specifying a point at the outlet side of the mesh (keyword *point_at_outlet*). Note that the vector defined by *vec_side* does not necessarily have to be perpendicular to the mesh face.

The following restrictions apply in case *vec_side* is specified: (i) the fix mesh/surface has to be planar, and (ii) the vector defined by *vec_side* may not lie inside the mesh plane.

The following restriction applies in case *point_at_outlet* is used: the *count* value has to be set to once.

The keyword *point_at_outlet* is especially useful in case a cylindrically-shaped surface is used. The

fix massflow/mesh command   231

point_at_outlet value should be on the cylinder axis in this case. If you like to track particles moving away from the cylinder axis, specify the *point_at_outlet* on the axis, and use the keyword *inside_out* to flip the direction.

When *count* = once, then each particle is only counted once, for *count* = multiple a particle contributes to number and mass count each time it crosses the mesh face. This can happen e.g. in the case of periodic boundary conditions or in re-circulating flow conditions.

The diameter, position and velocity of the particles can be written into a file using the *file* keyword, by specifying a filename.

If the *screen* keyword is used, output by this fix to the screen and logfile can be turned on or off as desired.

If the *delete_atoms* keyword is used then the particles passing through the mesh surface are deleted at the next re-neighboring step.

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

Information about this fix is written to **binary restart files** .

This fix computes a per-atom vector (the marker) which can be accessed by various **output commands**. The per-atom vector (i.e., the marker) can be accessed by dumps by using "f_massflow_ID", . This fix also computes a global vector of length 6. This vector can be accessed via "f_ID", where ID is the fix id. The first vector component is equal to the total mass which has crossed the mesh surface, the second vector component indicates the particle count. The third vector component is equal to the total mass which has crossed the mesh surface since the last output divived by the time since the last output (i.e., the mass flow rate), the fourth vector component indicates the particle count since the last output divived by the time since the last output (i.e., the number rate of particles). The fifth and sixth vector components are the deleted mass and the number of deleted particles. This vector can also be accessed by various **output commands**.

Restrictions:

Currently, this feature does not support multi-sphere particles.

Related commands:

compute nparticles/tracer/region

Default:

count = multiple, *inside_out* =false, *delete_atoms* = false

fix mesh/surface command

fix mesh/surface/planar command

Syntax:

```
fix ID group-ID mesh/surface file filename premesh_keywords premesh_values mesh_keywords mesh_val
fix ID group-ID mesh/surface/planar file filename premesh_keywords premesh_values mesh_keywords m
```

- ID, is documented in [fix](#) command.
- mesh/surface or mesh/surface/planar = style name of this fix command
- file = obligatory keyword
- filename = name of STL or VTK file containing the triangle mesh data
- zero or more premesh_keywords/premesh_value pairs may be appended
- premesh_keyword = *type* or *precision* or *heal* or *element_exclusion_list* or *verbose*

```
  type value = atom type (material type) of the wall imported from the STL file
    precision value = length mesh nodes this far away at maximum will be recognized as ident
    heal value = auto_remove_duplicates or no
    element_exclusion_list values = mode element_exlusion_file
      mode = read or write
      element_exlusion_file = name of file containing the elements to be excluded
    verbose value = yes or no
```

- zero or more mesh_keywords/mesh_value pairs may be appended
- mesh_keyword = *scale* or *move* or *rotate* or *temperature* or *mass_temperature*

```
  scale value = factor
      factor = factor to scale the mesh in x-, y-, and z-direction (double value)
    move values = mx my mz
      mx my mz = move the mesh by this extent in x-, y-, and z-direction (length units)
    rotate values = axis ax ay az angle ang
      axis = obligatory keyword
      ax, ay, az = axis vector for rotation (length units)
      angle = obligatory keyword
    temperature value = T0
      T0 = Temperature of the wall (temperature units)
    mass_temperature value = mt0
      mt0 = mass (mass units) assumed for temperature update in the frame of surface sphere/
```

- zero or more surface_keywords/surface_value pairs may be appended
- surface_keyword = *surface_vel* or *surface_ang_vel* or *curvature* or *curvature_tolerant*

```
  surface_vel values = vx vy vz
      vx vy vz = conveyor belt surface velocity  (velocity units)
    surface_ang_vel values = origin ox oy oz axis ax ay az omega om
      origin = mandatory keyword
      ox oy oz = origin of rotation (length units)
      axis = mandatory keyword
      ax ay az = axis vector for rotation (length units)
      omega = mandatory keyword
      om = rotaional velocity around specifyied axis (rad/time units)
    curvature value = c
      c = maximum angle between mesh faces belonging to the same surface (in degree)
    curvature_tolerant value = ct
      ct = yes or no
```

Examples:

```
fix cad all mesh/surface file mesh.stl type 1
```

Description:

This fix allows the import of triangual surfeace mesh wall geometry for granular simulations from ASCII STL files or legacy ASCII VTK files. Style *mesh/surface* is a general surface mesh, and *mesh/surface/planar* represents a planar mesh. *mesh/surface/planar* requires the mesh to consist of only 1 planar face.

Generall, you can apply scaling, offset and rotation to the imported mesh via keywords *scale*, *move*, *rotate*. Operations are applied in the order as they are specified. The group-ID defines which particles will "see" the mesh, in case it is used as a granular wall.

One fix represents one wall with a specific material, where the material is identified via keyword *type*. If multiple meshes with different materials are desired, the respective meshes must be imported with different fix mesh/surface commands.

With the *temperature* keyword, you can define a constant temperature for a mesh in conjunction with heat conduction via fix heat/gran. Note that the actual calculation of the heat transfer happens only if you use the mesh in conjunction with a granular wall, see fix wall/gran.

With the optional *surface_vel* keyword, you can specify the imported mesh as conveyor belt. The velocity direction for each mesh face is given by the projection of the conveyor belt velocity parallel to the mesh face, the velocity magnitude for each mesh face is equal to the conveyor belt velocity. This ensures the model makes sense also in case the mesh is curved. Likewise, the optional rotation model activated via keyword *surface_ang_vel* mimics rotational motion of the mesh (e.g. for modeling a shear cell)

Quality checks / error and warning messages:

LIGGGHTS(R)-WSU checks a couple of quality criteria upon loading a mesh. LIGGGHTS(R)-WSU tries to give you as much information about the issue as possible.

Warning messages:

- There should be no angle < 0.5 degrees in any element (soft limit for angles)
- All nodes should be within the simlation box

  If any of the obove rules is not fulfilled, a warning is generated. Keyword *verbose* controls if details about the warning are written to the screen.

  Error messages:
- The *curvature* must not be larger than any angle in any mesh element
- There must be no element with an angle < 0.181185 degrees in any element (hard limit for angles)
- The number of neighbor elements must be <= 5 for any element
- Mesh elements must not be duplicate
- Coplanar mesh elements that share an edge must not overlap
- Mesh elements must not leave the simulation domain (based on the center of the element)

  If any of the obove rules is not fulfilled, an error is generated and LIGGGHTS(R)-WSU halts. Error messages are always *verbose*. If LIGGGHTS(R)-WSU halts due to the last error, you might think about (a) changing the mesh import parameters (*scale*, *move*, *rotate*), (b) changing the mesh dynamics if a fix move/mesh is applied or using boundary m m m

  There are a couple of features that help you read / repair a mesh which can not be read correctly in the first place:
- *precision*

- *curvature*
- *heal*
- *element_exclusion_list*
- *curvature_tolerant*

The *precision* keyword specifies how far away mesh nodes can be at maximum to be recognized as identical. This is important for building the topology of the mesh and identify elements as neighbors. Normally, this should only be changed if the mesh file you are working with is suffering from precsion / round-off issues.

The *curvature* keyword lets you specify up to which angle between two triangles the triangles should be treated as belonging to the same surface (e.g. useful for bends). This angle is used to decide if (a) contact history is copied from one triangle to the other as the contact point proceeds and (b) if edge and corner interaction is calculated.

If LIGGGHTS(R)-WSU stalls because of duplicate elements, you can try setting *heal* to auto_remove_duplicates. LIGGGHTS(R)-WSU will then try to heal the geometry by removing duplicate elements.

With the optional *element_exclusion_list* keyword is used in mode 'write', LIGGGHTS(R)-WSU will write a list of elements which have more than the allowed 5 face neighbors per surface element to a file, as well as those elements which have an angle below 0.181185 degrees. The 'read' mode can then use this file and will skip the elements in the list. However, you can also manually write such a file to exclude elements you do not want to have included

IMPORTANT NOTE: The *element_exclusion_list write* model works in serial only. However, this is not a real restriction, since you can generate the exclusion lists in serial, but then read them to import the mesh into a parallel simulation

IMPORTANT NOTE: If you use the 'heal' or 'element_exclusion_list' keywords, you should check the changes to the geometry, e.g. by using a dump mesh/stl command.

The *curvature_toleran* keyword can simply turn off the check that the *curvature* must not be larger than any angle in any mesh element. This is typically not recommended, but can be used as a last resort measure.

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

This fix writes the STL data to binary restart files to be able to correctly resume the simulation in case the mesh is moved. None of the fix_modify options are relevant to this fix. No global scalar or vector or per-atom quantities are stored by this fix for access by various output commands. No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions:

To date, only ASCII STL and VTK files can be read (binary is not supported). In the current implementation, each processor allocates memory for the whole geometry, which may lead to memory issues for very large geometries . It is not supported to use both the moving mesh and the conveyor belt feature.

Related commands:

fix wall/gran

fix mesh/surface/planar command   235

Default: curvature = 0.256235 degrees, precision = 1e-8, verbose = no, heal = no

fix mesh/surface/stress command

Syntax:

```
fix ID group-ID mesh/surface/stress file filename premesh_keywords premesh_values mesh_keywords m
```

- ID, is documented in fix command, the group-ID is ignored for this command.
- mesh/surface/stress = style name of this fix command
- file filename premesh_keywords premesh_values mesh_keywords mesh_values surface_keyword surface_values are documented in fix mesh/surface.
- zero or more stress_keyword/value pairs may be appended
- stress_keyword = *stress* or *wear*

```
  stress value = on or off
    reference_point values = rx ry rz
      rx, ry, rz = coordinates of reference point
    wear value = finnie or off
```

Examples:

```
fix cad all mesh/surface/stress file mesh.stl type 1 wear finnie
```

Description:

This fix is identical to fix mesh/surface but additionally the pressure and shear force that the particles in the fix group exert on each triangle of the mesh is evaluated (which costs a bit of performance). Also, the total force and torque on the particle is calculated (see output info). The per-element forces can be dumped into VTK format using dump mesh/vtk.

With the optional *stress* keyword, stress tracking can be turned off is desired. The reference point for calculating the body torque can be controlled via the *referece_point* keyword. The optional *wear* keyword can activates a simple qualitative wear model (*finnie*) - for details on the model, see the seperate /doc/finnie-wear.pdf. The finnie constant k in Eqn. (4.23) has to be specified as follows:

```
fix id all property/global k_finnie peratomtypepair n_atomtypes value_11 value_12 .. value_21 val
```

```
    (value_ij=value for the finnie constant between atom type i and j; n_atomtypes is the number
```

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

This fix stores a global vector with 6 components for access by various output commands. The first 3 components are equal to the total force on the mesh, the last 3 components store the total torque on the body exerted by the particles. Other info see fix mesh.

Related commands:

fix mesh/surface fix wall/gran

Default:

stress = on reference_point = 0. 0. 0. wear = off

fix mesh/surface/stress/servo command

Syntax:

```
fix ID group-ID mesh/surface/stress/servo file filename premesh_keywords premesh_values mesh_keyw
```

- ID, is documented in fix command, the group-ID is ignored for this command.
- mesh/surface/stress/servo = style name of this fix command
- file filename premesh_keywords premesh_values mesh_keywords mesh_values surface_keyword surface_values stress_keywords stress_values are documented in fix mesh/surface/stress.
- zero or more servo_keyword/value pairs may be appended servo keywords = *com* (obligatory) or *dim* (obligatory) or *ctrlPV* (obligatory) or *vel_max* (obligatory) or *kp* or *ki* or *kd*

```
com values = x, y, z
    x, y, z = coordinates of the center of mass of the body (distance units)
ctrlPV values = force or torque
    force = use force as controll process value, i.e. control force
    torque = use torque as controll process value, i.e. control torque
axis args = x y z
    x y z = vector direction to apply the controlled mesh motion
    x or y or z can be a variable (see below)
target_val values = val
    val = target value for the controller (force units or torque units, depending on ctrlP
vel_max values = v
    v = maximum velocity magnitude for servo wall (velocity units)
kp values = k
    k = proportional constant for PID controller
ki values = k
    k = integral constant for PID controller
kd values = k
    k = differential constant for PID controller
mode values = auto
    auto = use alternative controller algorithm
ratio values = dr
    dr = constant for the alternative controller approach (mode = auto)
```

Examples:

```
fix servo all mesh/surface/stress/servo file plate.stl type 1 com 0. 0. 0. ctrlPV force axis 0. 0
fix servo all mesh/surface/stress/servo file stirrer.stl type 1 com 0. 0. 0. ctrlPV torque axis 0
```

Description:

This fix implements the functionality of fix mesh/surface/stress but it additionally assumes the mesh being a servo wall that compacts a particle packing until either a total force (for *ctrlPV* = force) or a total torque (for *ctrlPV* = torque) is acting on the mesh. The target value is defined via keyword *target_val*. The servo can act in any dimension (as specified by the *axis* keyword). Only the direction of the axis is important; it's length is ignored. A negative value for *target_val* leads to a wall motion towards negative *axis*-direction and vice versa. The user has to specify the center of mass (keyword *com*) and the maximum velocity allowed for the servo wall by keyword *vel_max*. Note that *vel_max* < skin /(2* timestep) is required.

The controller itself is a proportional-integral-derivative (PID) controller which is controlled by 3 constants *kp*, *ki*, *kd*:

output = kp * error + ki * errorsum + kd * errorchange

where 'error' is the current deviation of the controll process value to the target value, 'errorsum' is the time integration (sum) of the error values and 'errorchange' its derivative. The controller also includes an "anti-wind-up scheme" which prohibits accumulation of erroneous controller output caused by the integral part due to unavoidable long-lasting deviations.

By using the keyword *mode* = auto an alternative controller approach is applied. It is a pure proportional controller with gain scheduling. In the absence of neighbour particles the servo wall may move with maximum velocity (defined by *vel_max*). Otherwise, the maximum wall velocity is defined by *ratio* * min(radius) / dt. Approaching *target_val* the maximum velocity decreases to 0.1 * *ratio* * min(radius) / dt.

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

This fix stores a global vector with 9 components for access by various <u>output commands</u>. The first 3 components are equal to the total force on the mesh, the next 3 components store the total torque on the mesh. The last 3 components output the wall position. Furthermore, this fix writes the state of the servo wall to binary restart files so that a simulation can continue correctly. This fix supports <u>fix_modify</u> with option *integrate* = 'start' or 'stop' to start or stop the servo wall integration inbetween two runs. This fix also supports <u>fix_modify</u> with option *target_val* = val to change the target value inbetween two runs. This fix also supports <u>fix_modify</u> with option *ctrlParam* = kp ki kd to change the controller params inbetween two runs.

Restrictions:

When using this fix, along with scaling or rotate the body, all the servo_keyword/value pairs have to represent the state after scaling/rotation. Mesh elements may not be deleted in case due to leaving the simulation box for a fixed boundary. In this case, an error is generated. See <u>boundary</u> command for details. This fix can not be used in conjunction with another fix that manipulates mesh geometry, such as a <u>fix move/mesh</u>

Related commands:

<u>fix mesh/surface/stress</u>, <u>fix wall/gran</u>

Default:

kp = 1e-2, ki = 0, kd = 0

fix_modify command

Syntax:

```
fix_modify fix-ID keyword value ...
```

- fix-ID = ID of the fix to modify
- one or more keyword/value pairs may be appended
- keyword = *temp* or *press* or *energy*

```
  temp value = compute ID that calculates a temperature
    press value = compute ID that calculates a pressure
    energy value = yes or no
```

Examples:

```
fix_modify 3 temp myTemp press myPress
fix_modify 1 energy yes
```

Description:

Modify one or more parameters of a previously defined fix. Only specific fix styles support specific parameters. See the doc pages for individual fix commands for info on which ones support which fix_modify parameters.

The *temp* keyword is used to determine how a fix computes temperature. The specified compute ID must have been previously defined by the user via the [compute](#) command and it must be a style of compute that calculates a temperature. All fixes that compute temperatures define their own compute by default, as described in their documentation. Thus this option allows the user to override the default method for computing T.

The *press* keyword is used to determine how a fix computes pressure. The specified compute ID must have been previously defined by the user via the [compute](#) command and it must be a style of compute that calculates a pressure. All fixes that compute pressures define their own compute by default, as described in their documentation. Thus this option allows the user to override the default method for computing P.

For fixes that calculate a contribution to the potential energy of the system, the *energy* keyword will include that contribution in thermodynamic output of potential energy. See the [thermo_style](#) command for info on how potential energy is output. The contribution by itself can be printed by using the keyword f_ID in the thermo_style custom command, where ID is the fix-ID of the appropriate fix. Note that you must use this setting for a fix if you are using it when performing an [energy minimization](#) and if you want the energy and forces it produces to be part of the optimization criteria.

Restrictions: none

Related commands:

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

Default:

The option defaults are temp = ID defined by fix, press = ID defined by fix, energy = no.

---

fix momentum command

Syntax:

```
fix ID group-ID momentum N keyword values ...
```

- ID, group-ID are documented in [fix](#) command
- momentum = style name of this fix command
- N = adjust the momentum every this many timesteps one or more keyword/value pairs may be appended
- keyword = *linear* or *angular*

```
  linear values = xflag yflag zflag
      xflag,yflag,zflag = 0/1 to exclude/include each dimension
    angular values = none
```

Examples:

```
fix 1 all momentum 1 linear 1 1 0
fix 1 all momentum 100 linear 1 1 1 angular
```

Description:

Zero the linear and/or angular momentum of the group of atoms every N timesteps by adjusting the velocities of the atoms. One (or both) of the *linear* or *angular* keywords must be specified.

If the *linear* keyword is used, the linear momentum is zeroed by subtracting the center-of-mass velocity of the group from each atom. This does not change the relative velocity of any pair of atoms. One or more dimensions can be excluded from this operation by setting the corresponding flag to 0.

If the *angular* keyword is used, the angular momentum is zeroed by subtracting a rotational component from each atom.

This command can be used to insure the entire collection of atoms (or a subset of them) does not drift or rotate during the simulation due to random perturbations (e.g. [fix langevin](#) thermostatting).

Note that the [velocity](#) command can be used to create initial velocities with zero aggregate linear and/or angular momentum.

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

No information about this fix is written to [binary restart files](#). None of the [fix_modify](#) options are relevant to this fix. No global or per-atom quantities are stored by this fix for access by various [output commands](#). No parameter of this fix can be used with the *start/stop* keywords of the [run](#) command. This fix is not invoked during [energy minimization](#).

Restrictions: none

Related commands:

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

Default: none

fix move command

Syntax:

```
fix ID group-ID move style args keyword values ...
```

- ID, group-ID are documented in [fix](fix) command
- move = style name of this fix command
- style = *linear* or *wiggle* or *rotate* or *variable*

```
  linear args = Vx Vy Vz
     Vx,Vy,Vz = components of velocity vector (velocity units), any component can be specif
    wiggle args = Ax Ay Az period
     Ax,Ay,Az = components of amplitude vector (distance units), any component can be speci
     period = period of oscillation (time units)
   rotate args = Px Py Pz Rx Ry Rz period
     Px,Py,Pz = origin point of axis of rotation (distance units)
     Rx,Ry,Rz = axis of rotation vector
     period = period of rotation (time units)
   variable args = v_dx v_dy v_dz v_vx v_vy v_vz
     v_dx,v_dy,v_dz = 3 variable names that calculate x,y,z displacement as function of tim
     v_vx,v_vy,v_vz = 3 variable names that calculate x,y,z velocity as function of time, a
```

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

```
  units value = box or lattice
```

Examples:

```
fix 1 boundary move wiggle 3.0 0.0 0.0 1.0 units box
fix 2 boundary move rotate 0.0 0.0 0.0 0.0 0.0 1.0 5.0
fix 2 boundary move variable v_myx v_myy NULL v_VX v_VY NULL
```

Description:

Perform updates of position and velocity for atoms in the group each timestep using the specified settings or formulas, without regard to forces on the atoms. This can be useful for boundary or other atoms, whose movement can influence nearby atoms.

IMPORTANT NOTE: The atoms affected by this fix should not normally be time integrated by other fixes (e.g. [fix nve](fix nve), [fix nvt](fix nvt)), since that will change their positions and velocities twice.

IMPORTANT NOTE: As atoms move due to this fix, they will pass thru periodic boundaries and be remapped to the other side of the simulation box, just as they would during normal time integration (e.g. via the [fix nve](fix nve) command). It is up to you to decide whether periodic boundaries are appropriate with the kind of atom motion you are prescribing with this fix.

IMPORTANT NOTE: As dicsussed below, atoms are moved relative to their initial position at the time the fix is specified. These initial coordinates are stored by the fix in "unwrapped" form, by using the image flags associated with each atom. See the dump custom command for a discussion of "unwrapped" coordinates. See the Atoms section of the read_data command for a discussion of image flags and how they are set for each atom. You can reset the image flags (e.g. to 0) before invoking this fix by using the set image command.

The *linear* style moves atoms at a constant velocity, so that their position $X = (x,y,z)$ as a function of time is given in vector notation as

```
X(t) = X0 + V * delta
```

where $X0 = (x0,y0,z0)$ is their position at the time the fix is specified, V is the specified velocity vector with components (Vx,Vy,Vz), and *delta* is the time elapsed since the fix was specified. This style also sets the velocity of each atom to $V = (Vx,Vy,Vz)$. If any of the velocity components is specified as NULL, then the position and velocity of that component is time integrated the same as the fix nve command would perform, using the corresponding force component on the atom.

Note that the *linear* style is identical to using the *variable* style with an equal-style variable that uses the vdisplace() function. E.g.

```
variable V equal 10.0
variable x equal vdisplace(0.0,$V)
fix 1 boundary move variable v_x NULL NULL v_V NULL NULL
```

The *wiggle* style moves atoms in an oscillatory fashion, so that their position $X = (x,y,z)$ as a function of time is given in vector notation as

```
X(t) = X0 + A sin(omega*delta)
```

where $X0 = (x0,y0,z0)$ is their position at the time the fix is specified, A is the specified amplitude vector with components (Ax,Ay,Az), *omega* is 2 PI / *period*, and *delta* is the time elapsed since the fix was specified. This style also sets the velocity of each atom to the time derivative of this expression. If any of the amplitude components is specified as NULL, then the position and velocity of that component is time integrated the same as the fix nve command would perform, using the corresponding force component on the atom.

Note that the *wiggle* style is identical to using the *variable* style with equal-style variables that use the swiggle() and cwiggle() functions. E.g.

```
variable A equal 10.0
variable T equal 5.0
variable omega equal 2.0*PI/$T
variable x equal swiggle(0.0,$A,$T)
variable v equal v_omega*($A-cwiggle(0.0,$A,$T))
fix 1 boundary move variable v_x NULL NULL v_v NULL NULL
```

The *rotate* style rotates atoms around a rotation axis $R = (Rx,Ry,Rz)$ that goes thru a point $P = (Px,Py,Pz)$. The *period* of the rotation is also specified. This style also sets the velocity of each atom to (omega cross Rperp) where omega is its angular velocity around the rotation axis and Rperp is a perpendicular vector from the rotation axis to the atom. If the defined atom_style assigns an angular velocity to each atom, then each atom's angular velocity is also set to omega. Note that the direction of rotation for the atoms around the rotation axis is consistent with the right-hand rule: if your right-hand's thumb points along R, then your fingers wrap around the axis in the direction of rotation.

The *variable* style allows the position and velocity components of each atom to be set by formulas specified via the variable command. Each of the 6 variables is specified as an argument to the fix as v_name, where name is the variable name that is defined elsewhere in the input script.

Each variable must be of either the *equal* or *atom* style. *Equal*-style variables compute a single numeric quantity, that can be a function of the timestep as well as of other simulation values. *Atom*-style variables compute a numeric quantity for each atom, that can be a function per-atom quantities, such as the atom's position, as well as of the timestep and other simulation values. Note that this fix stores the original coordinates of each atom (see note below) so that per-atom quantity can be used in an atom-style variable

fix move command   244

formula. See the [variable](#) command for details.

The first 3 variables (v_dx,v_dy,v_dz) specified for the *variable* style are used to calculate a displacement from the atom's original position at the time the fix was specified. The second 3 variables (v_vx,v_vy,v_vz) specified are used to compute a velocity for each atom.

Any of the 6 variables can be specified as NULL. If both the displacement and velocity variables for a particular x,y,z component are specified as NULL, then the position and velocity of that component is time integrated the same as the [fix nve](#) command would perform, using the corresponding force component on the atom. If only the velocity variable for a component is specified as NULL, then the displacement variable will be used to set the position of the atom, and its velocity component will not be changed. If only the displacement variable for a component is specified as NULL, then the velocity variable will be used to set the velocity of the atom, and the position of the atom will be time integrated using that velocity.

The *units* keyword determines the meaning of the distance units used to define the *linear* velocity and *wiggle* amplitude and *rotate* origin. This setting is ignored for the *variable* style. A *box* value selects standard units as defined by the [units](#) command, e.g. velocity in Angstroms/fmsec and amplitude and position in Angstroms for units = real. A *lattice* value means the velocity units are in lattice spacings per time and the amplitude and position are in lattice spacings. The [lattice](#) command must have been previously used to define the lattice spacing. Each of these 3 quantities may be dependent on the x,y,z dimension, since the lattice spacings can be different in x,y,z.

For [rRESPA time integration](#), this fix adjusts the position and velocity of atoms on the outermost rRESPA level.

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

This fix writes the original coordinates of moving atoms to [binary restart files](#), as well as the initial timestep, so that the motion can be continuous in a restarted simulation. See the [read_restart](#) command for info on how to re-specify a fix in an input script that reads a restart file, so that the operation of the fix continues in an uninterrupted fashion.

IMPORTANNT NOTE: Because the move positions are a function of the current timestep and the initial timestep, you cannot reset the timestep to a different value after reading a restart file, if you expect a fix move command to work in an uninterrupted fashion.

None of the [fix_modify](#) options are relevant to this fix.

This fix produces a per-atom array which can be accessed by various [output commands](#). The number of columns for each atom is 3, and the columns store the original unwrapped x,y,z coords of each atom. The per-atom values can be accessed on any timestep.

No parameter of this fix can be used with the *start/stop* keywords of the [run](#) command. This fix is not invoked during [energy minimization](#).

Restrictions: none

Related commands:

[fix nve](#), [displace_atoms](#)

Default: none

The option default is units = box.

fix move/mesh command

Syntax:

```
fix ID group-ID move/mesh mesh mesh-ID style args keyword values ...
```

- ID is documented in fix command, group-ID is ignored
- move/mesh = style name of this fix command
- mesh = obligatory keyword
- mesh-ID = ID for the fix mesh that the fix move/mesh is applied to
- style = *linear* or *linear/variable* or *wiggle* or *riggle* or *rotate* or *rotate/variable* or *viblin* or *vibrot*

```
  linear args = Vx Vy Vz
      Vx,Vy,Vz = components of velocity vector (velocity units)
    linear/variable args = var_Vx var_Vy var_Vz
      var_Vx,var_Vy,var_Vz = variables specifying components of velocity vector (velocity un
    wiggle args = amplitude Ax Ay Az period per
      amplitude = obligatory keyword
      Ax,Ay,Az = components of amplitude vector (distance units)
      period = obligatory keyword
      per = period of oscillation (time units)
    riggle args = origin Px Py Pz axis ax ay az period per amplitude ampl
      origin = obligatory keyword
      Px,Py,Pz = origin point of axis of rotation (distance units)
      axis = obligatory keyword
      ax,ay,az = axis of rotation vector (distance units)
      period = obligatory keyword
      per = period of rotation (time units)#
      amplitude = obligatory keyword
      ampl = amplitude of riggle movement (grad)
    rotate args = origin Px Py Pz axis ax ay az period per
      origin = obligatory keyword
      Px,Py,Pz = origin point of axis of rotation (distance units)
      ax,ay,az = axis of rotation vector (distance units)
      period = obligatory keyword
      per = period of rotation (time units)
    rotate/variable args = origin Px Py Pz axis ax ay az omega var_omega
      origin = obligatory keyword
      Px,Py,Pz = origin point of axis of rotation (distance units)
      ax,ay,az = axis of rotation vector (distance units)
      omega = obligatory keyword
      var_omega = variable specifying angular velocity (rad / time units)
    viblin args = axis ax ay az order n amplitude C1 ... Cn phase p1 ... pn period per
      axis = obligatory keyword
      ax,ay,az = components of moving direction vector (distance units)(origin 0 0 0)
      order= obligatory keyword
      n= order of trigonometric series n (from 1 to 30)
      amplitude = obligatory keyword
      C1, ..., Cn = amplitude (distance units)
      phase = obligatory keyword
      p1, ...,pn = phase of functionterm (rad) (number of terms is equivalent to order n)
      period = obligatory keyword
      per_1, ..., per_n = period of rotation (time units)
    vibrot args = origin Px Py Pz axis ax ay az order n amplitude C1 ... Cn phase p1 ... pn 
      origin = obligatory keyword
      Px,Py,Pz = origin point of axis of rotation (distance units)
      axis = obligatory keyword
      ax,ay,az = axis of rotation vector
      order= obligatory keyword
      n= order of trigonometric series (from 1 to 30)
```

```
        amplitude = obligatory keyword
        C1, ..., Cn = amplitude (rad)
        phase = obligatory keyword
        p1, ...,pn = phase of functionterm (rad) (number of terms is equivalent to order n)
        period = obligatory keyword
        per_1, ..., per_n = period of rotation (time units)
```

Examples:

```
fix move all move/mesh mesh cad1 wiggle amplitude -0.1 0. 0. period 0.02
fix move all move/mesh mesh cad1 rotate origin 0. 0. 0. axis  0. 0. 1. period 0.05
fix move all move/mesh mesh cad1 linear 5. 5. 0.
fix move all move/mesh mesh cad1 viblin axis 0. 0. 1 order 5 amplitude 0.4 0.1 0.3 0.1 0.1 phase
fix move all move/mesh mesh cad1 vibrot origin 0. 0. 0 axis 0. 0. 1 order 2 amplitude 0.4 0.1  ph
```

Description:

Perform updates of position and velocity for mesh elements which are part of the fix_mesh_surface with ID *mesh-ID* using the specified settings or formulas. The fix group is ignored for this command.

The *linear* style moves mesh elements at a constant velocity, so that their position $X = (x,y,z)$ as a function of time is given in vector notation as

```
X(t) = X0 + V * delta
```

where $X0 = (x0,y0,z0)$ is their position at the time the fix is specified, *V* is the specified velocity vector with components (Vx,Vy,Vz), and *delta* is the time elapsed since the fix was specified. This style also sets the velocity of each atom to $V = (Vx,Vy,Vz)$.

The *linear/variable* style does the same as the *linear* style, but uses three variables so that the velocity can be time-dependant.

The *wiggle* style moves atoms in an oscillatory fashion, so that their position $X = (x,y,z)$ as a function of time is given in vector notation as

```
X(t) = X0 + A sin(omega*delta)
```

where $X0 = (x0,y0,z0)$ is their position at the time the fix is specified, *A* is the specified amplitude vector with components (Ax,Ay,Az), *omega* is 2 PI / *period*, and *delta* is the time elapsed since the fix was specified. This style also sets the velocity of each element to the time derivative of this expression.

The *rotate* style rotates around a rotation axis $R = (Rx,Ry,Rz)$ that goes thru a point $P = (Px,Py,Pz)$. The *period* of the rotation is also specified. This style also sets the velocity of each element to (omega cross Rperp) where omega is its angular velocity around the rotation axis and Rperp is a perpendicular vector from the rotation axis to the atom. Note that the direction of rotation around the rotation axis is consistent with the right-hand rule: if your right-hand's thumb points along *R*, then your fingers wrap around the axis in the direction of rotation.

The *rotate/variable* style does the same as the *rotate* style, but uses a variable for the angular velocity so that the angular velocity can be time-dependant. IMPORTANT NOTE: style *rotate* takes the period of the rotation as input, *rotate/variable* takes angular velocity as input.

The *riggle* style imposes an oscillatory rotation around a rotation axis $R = (Rx,Ry,Rz)$ that goes thru a point $P = (Px,Py,Pz)$. The *period* of the oscillation is specified as well as the *amplitude* in grad (Â°). This style also sets the velocity of each element accordingly.

The *viblin* style moves meshes in an oscillatory fashion with an vibration function of higher order, so that their position $X = (x,y,z)$ as a function of time is given in vector notation as

$$X(t) = X_0 + D \cdot \sum_{n=1}^{k} C_n \cdot \cos(omega_n \cdot delta + phase_n)$$

where *X0* = (x0,y0,z0) is their position at the time the fix is specified, *n* represents the order of the trigonometric series, *Cn* is the specified amplitude along the direction given by *axis* = (ax,ay,az). The vector *D* is the unit vector of *axis*. The angular velocity *omega n* is 2 PI / *period n*, and *delta* is the time elapsed since the fix was specified. This style also sets the velocity of each element to the time derivative of this expression.

The *vibrot* style generates an oscillatory rotation around a rotation *axis* = (ax,ay,az) that goes thru a point *origin* = (Px,Py,Pz). The *period* of the oscillation is used to calculate omega, the amplitudes *Cn* and the phase phase *n* are given in rad. The change of rotation angle per time *gamma(t)* is described by trigonometric series of order *n*. The formula for this change is

$$gamma(t) = \sum_{n=1}^{k} C_n \cdot \cos(omega_n \cdot delta + phase_n)$$

This style also sets the velocity of each element accordingly

NOTE: If a dangerous tri neighbor list build is detected, this may be due to the fact that the geometry is moved too close to a region where particle insertion is taking place so that initial interpenetration happens when the particles are inserted.

NOTE: When moving a mesh element, not only the node positions are moved but also a couple of other vectors. So moving one mesh element is more costly as one particle.

Superposition of multiple fix move/mesh commands:

It is possible to superpose multiple fix move/mesh commands. In this case, the reference frame for the second move command moves along as the mesh is moved by the first move command etc. E.g. for style *rotate*, the origin of the rotation axis would be in local reference frame.

Example: A mesh should rotate around a central axis and additionally revolve around its center of mass. The first move command should be the rotation around the central axis, the second move command the revolution around the center of mass of the mesh.

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

This fix writes the original coordinates of moving elements to binary restart files, so that the motion can be continuous in a restarted simulation. See the read_restart command for info on how to re-specify a fix in an input script that reads a restart file, so that the operation of the fix continues in an uninterrupted fashion.

None of the fix_modify options are relevant to this fix.

No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions:

If multiple fix move/mesh movements are superposed onto one mesh, they have to be deleted in reverse order of their creation Mesh elements may not be deleted in case due to leaving the simulation box for a fixed boundary. In this case, an error is generated. See boundary command for details.

Related commands:

[fix mesh/surface](fix_mesh_surface)

Default: none

fix multisphere command

Syntax:

```
fix ID group-ID multisphere
```

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

Examples:

fix ms all multisphere

Description:

Treat one or more sets of atoms as independent rigid bodies. This means that each timestep the total force and torque on each rigid body is computed as the sum of the forces and torques on its constituent particles and the coordinates, velocities, and orientations of the atoms in each body are updated so that the body moves and rotates as a single entity. The integration that is performed by this command is euqivalent to [fix_rigid](#). However, the following implementation details are different:

(1) Body data held by this fix is distributed across all processes, yielding better parallel scalabilty.

(2) Bodies (particle clumps) can be added/inserted via [fix_insert_stream](#), [fix_insert_rate_region](#) or [fix_insert_pack](#) and are automatically added to this fix.

(3) A body is removed from the simulation domain if any of it's particles is removed from the simulation according to the [boundary](#) settings or by any command that deleted particles (e.g. [delete_atoms](#)).

(4) By using this fix, gravity ([fix_gravity](#)) will be handled correctly for for overlapping particle clumps.

(5) This fix internally performs a [neigh_modify](#) exclude command so that particles belonging to the same rigid body are excluded from the neighborlist build.

IMPORTANT NOTE: You should not update the atoms in rigid bodies via other time-integration fixes (e.g. nve, nvt, npt), or you will be integrating their motion more than once each timestep.

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

No information about this fix is written to [binary restart files](#). This means you can currently not restart a simulation using multisphere particles.

None of the [fix_modify](#) options are relevant to this fix. No global scalar or vector or per-atom quantities are stored by this fix for access by various [output commands](#). No parameter of this fix can be used with the *start/stop* keywords of the [run](#) command. This fix is not invoked during [energy minimization](#).

Restrictions:

Only one fix multisphere at a time is allowed. Heat transfer simulations are not possible when using this fix (e.g. [fix heat/gran/conduction](#))

The PUBLIC version does not support (i) parallel computation, (ii) restart of multisphere simulations

IMPORTANT NOTE: All fixes or computes gathering statistical output (such as e.g. compute com or fix ave/time operate on a per-sphere rather than on a per-body basis.

Currently, using fix multisphere requires newton = off and dimension = 3.

Related commands:

fix_rigid fix_particletemplate_sphere neigh_modify

Default: none

fix nve/asphere command

Syntax:

```
fix ID group-ID nve/asphere
```

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

Examples:

```
fix 1 all nve/asphere
```

Description:

Perform constant NVE integration to update position, velocity, orientation, and angular velocity for aspherical particles in the group each timestep. V is volume; E is energy. This creates a system trajectory consistent with the microcanonical ensemble.

This fix differs from the [fix nve](#) command, which assumes point particles and only updates their position and velocity.

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

No information about this fix is written to [binary restart files](#). None of the [fix_modify](#) options are relevant to this fix. No global or per-atom quantities are stored by this fix for access by various [output commands](#). No parameter of this fix can be used with the *start/stop* keywords of the [run](#) command. This fix is not invoked during [energy minimization](#).

Restrictions:

This fix is part of the ASPHERE package. It is only enabled if LIGGGHTS(R)-WSU was built with that package. See the [Making LIGGGHTS(R)-WSU](#) section for more info.

This fix requires that atoms store torque and angular momementum and a quaternion as defined by the [atom_style ellipsoid](#) command.

All particles in the group must be finite-size. They cannot be point particles, but they can be aspherical or spherical as defined by their shape attribute.

Related commands:

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

Default: none

fix nve/asphere/noforce command

Syntax:

```
fix ID group-ID nve/asphere/noforce
```

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

Examples:

fix 1 all nve/asphere/noforce

Description:

Perform updates of position and orientation, but not velocity or angular momentum for atoms in the group each timestep. In other words, the force and torque on the atoms is ignored and their velocity and angular momentum are not updated. The atom velocities and angularm momenta are used to update their positions and orientation.

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

No information about this fix is written to [binary restart files](#). None of the [fix_modify](#) options are relevant to this fix. No global or per-atom quantities are stored by this fix for access by various [output commands](#). No parameter of this fix can be used with the *start/stop* keywords of the [run](#) command. This fix is not invoked during [energy minimization](#).

Restrictions:

This fix is part of the ASPHERE package. It is only enabled if LIGGGHTS(R)-WSU was built with that package. See the [Making LIGGGHTS(R)-WSU](#) section for more info.

This fix requires that atoms store torque and angular momementum and a quaternion as defined by the [atom_style ellipsoid](#) command.

All particles in the group must be finite-size. They cannot be point particles, but they can be aspherical or spherical as defined by their shape attribute.

Related commands:

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

Default: none

fix nve command

Syntax:

```
fix ID group-ID nve
```

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

Examples:

```
fix 1 all nve
```

Description:

Perform constant NVE integration to update position and velocity for atoms in the group each timestep. V is volume; E is energy. This creates a system trajectory consistent with the microcanonical ensemble.

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

No information about this fix is written to [binary restart files](#). None of the [fix_modify](#) options are relevant to this fix. No global or per-atom quantities are stored by this fix for access by various [output commands](#). No parameter of this fix can be used with the *start/stop* keywords of the [run](#) command. This fix is not invoked during [energy minimization](#).

Restrictions: none

Related commands: none

Default: none

fix nve/limit command

Syntax:

```
fix ID group-ID nve/limit limitstyle xmax
```

- ID, group-ID are documented in [fix](#) command
- nve/limit = style name of this fix command
- limitstyle = absolute or radius_ratio
- xmax = maximum distance an atom can move in one timestep (distance units or relative to atom radius)

Examples:

```
fix 1 all nve/limit absolute 0.1
```

Description:

Perform constant NVE updates of position and velocity for atoms in the group each timestep. A limit is imposed on the maximum distance an atom can move in one timestep. This is useful when starting a simulation with a configuration containing highly overlapped atoms. Normally this would generate huge forces which would blow atoms out of the simulation box, causing LIGGGHTS(R)-WSU to stop with an error.

Using this fix can overcome that problem. Forces on atoms must still be computable (which typically means 2 atoms must have a separation distance > 0.0). But large velocities generated by large forces are reset to a value that corresponds to a displacement of length *xmax* in a single timestep. *Xmax* is specified in distance units; see the [units](#) command for details. The value of *xmax* should be consistent with the neighbor skin distance and the frequency of neighbor list re-building, so that pairwise interactions are not missed on successive timesteps as atoms move. See the [neighbor](#) and [neigh_modify](#) commands for details.

If limitstyle *absolute* is used, *xmax* is applied directly. If limitstyle *radius_ratio* is used, a maxmimum distance per step of *xmax**radius is applied for each atom. This can be useful for the simulation of poly-disperse systems. Note that this option requires the atom radius to be stored by using an appropriate atom style.

Note that if a velocity reset occurs the integrator will not conserve energy. On steps where no velocity resets occur, this integrator is exactly like the [fix nve](#) command. Since forces are unaltered, pressures computed by thermodynamic output will still be very large for overlapped configurations.

IMPORTANT NOTE: You should not use [fix shake](#) in conjunction with this fix. That is because fix shake applies contraint forces based on the predicted postitions of atoms after the next timestep. It has no way of knowing the timestep may change due to this fix, which will cause the constraint forces to be invalid. A better strategy is to turn off fix shake when performing initial dynamics that need this fix, then turn fix shake on when doing normal dynamics with a fixed-size timestep.

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

No information about this fix is written to [binary restart files](#). None of the [fix_modify](#) options are relevant to this fix.

This fix computes a global scalar which can be accessed by various **output commands**. The scalar is the count of how many updates of atom's velocity/position were limited by the maximum distance criterion. This should be roughly the number of atoms so affected, except that updates occur at both the beginning and end of a timestep in a velocity Verlet timestepping algorithm. This is a cumulative quantity for the current run, but is re-initialized to zero each time a run is performed. The scalar value calculated by this fix is "extensive".

No parameter of this fix can be used with the *start/stop* keywords of the **run** command. This fix is not invoked during **energy minimization**.

Restrictions: none

Related commands:

fix nve, fix nve/noforce, pair_style soft

Default: none

fix nve/line command

Syntax:

```
fix ID group-ID nve/line
```

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

Examples:

```
fix 1 all nve/line
```

Description:

Perform constant NVE integration to update position, velocity, orientation, and angular velocity for line segment particles in the group each timestep. V is volume; E is energy. This creates a system trajectory consistent with the microcanonical ensemble. See [Section_howto 14](#) of the manual for an overview of using line segment particles.

This fix differs from the [fix nve](#) command, which assumes point particles and only updates their position and velocity.

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

No information about this fix is written to [binary restart files](#). None of the [fix_modify](#) options are relevant to this fix. No global or per-atom quantities are stored by this fix for access by various [output commands](#). No parameter of this fix can be used with the *start/stop* keywords of the [run](#) command. This fix is not invoked during [energy minimization](#).

Restrictions:

This fix is part of the ASPHERE package. It is only enabled if LIGGGHTS(R)-WSU was built with that package. See the [Making LIGGGHTS(R)-WSU](#) section for more info.

This fix requires that particles be line segments as defined by the [atom_style line](#) command.

Related commands:

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

Default: none

fix nve/noforce command

Syntax:

```
fix ID group-ID nve
```

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

Examples:

```
fix 3 wall nve/noforce
```

Description:

Perform updates of position, but not velocity for atoms in the group each timestep. In other words, the force on the atoms is ignored and their velocity is not updated. The atom velocities are used to update their positions.

This can be useful for wall atoms, when you set their velocities, and want the wall to move (or stay stationary) in a prescribed fashion.

This can also be accomplished via the [fix setforce](#) command, but with fix nve/noforce, the forces on the wall atoms are unchanged, and can thus be printed by the [dump](#) command or queried with an equal-style [variable](#) that uses the fcm() group function to compute the total force on the group of atoms.

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

No information about this fix is written to [binary restart files](#). None of the [fix_modify](#) options are relevant to this fix. No global or per-atom quantities are stored by this fix for access by various [output commands](#). No parameter of this fix can be used with the *start/stop* keywords of the [run](#) command. This fix is not invoked during [energy minimization](#).

Restrictions: none

Related commands:

[fix nve](#)

Default: none

fix nve/sphere command

Syntax:

```
fix ID group-ID nve/sphere
```

- ID, group-ID are documented in [fix](#) command
- nve/sphere = style name of this fix command
- zero or more keyword/value pairs may be appended
- keyword = *update*

```
  update value = dipole
      dipole = update orientation of dipole moment during integration
```

Examples:

```
fix 1 all nve/sphere
fix 1 all nve/sphere update dipole
```

Description:

Perform constant NVE integration to update position, velocity, and angular velocity for finite-size spherical particles in the group each timestep. V is volume; E is energy. This creates a system trajectory consistent with the microcanonical ensemble.

This fix differs from the [fix nve](#) command, which assumes point particles and only updates their position and velocity.

If the *update* keyword is used with the *dipole* value, then the orientation of the dipole moment of each particle is also updated during the time integration. This option should be used for models where a dipole moment is assigned to particles via use of the [atom_style dipole](#) command.

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

No information about this fix is written to [binary restart files](#). None of the [fix_modify](#) options are relevant to this fix. No global or per-atom quantities are stored by this fix for access by various [output commands](#). No parameter of this fix can be used with the *start/stop* keywords of the [run](#) command. This fix is not invoked during [energy minimization](#).

Restrictions:

This fix requires that atoms store torque and angular velocity (omega) and a radius as defined by the [atom_style sphere](#) command. If the *dipole* keyword is used, then they must also store a dipole moment as defined by the [atom_style dipole](#) command.

All particles in the group must be finite-size spheres. They cannot be point particles.

Related commands:

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

Default: none

fix particledistribution/discrete command

fix particledistribution/discrete/massbased command

fix particledistribution/discrete/numberbased command

Syntax:

```
fix ID group-ID particledistribution/discrete seed ntemp t_id t_weight ...
```

- ID, group-ID are documented in fix command
- particledistribution/discrete or particledistribution/discrete/massbased or particledistribution/discrete/numberbased = style name of this fix command
- seed = random number generator seed (integer value)
- ntemp = number of particle templates to be used in this command
- zero or more *t_id*/*t_weight* pairs are appended, number of pairs must match ntemp

```
    t_id = ID of a fix of type particletemplate/sphere
      t_weight = mass % or number % for this template in the distribution
```

Examples:

```
fix pdd1 all particledistribution/discrete 6778  1 pts1 1.0
fix pdd1 all particledistribution/discrete 1239  2 pts1 0.3 pts2 0.7
```

Description:

Define a discrete particle distribution that defines a discrete particle distribution to be inserted by a fix insert/stream, fix insert/pack, fix insert/rate/region or a related command. It takes several templates of type fix_particletemplate_sphere as input, which define the properties of the single particles (such as radius, density that are part of the distribution. The pairs of IDs and weights for the templates (*t_id* / *t_weight*) define the distribution. Please note that the number of pairs must match *ntemp*, but can be arbitrtarily large, so that any type of particle size distribution can be discretized.

For style *particledistribution/discrete* or *particledistribution/discrete/massbased* the weight of each template within the distribution is interpreted as mass-%, for style *particledistribution/discrete/numberbased* the weight is interpreted as number-%. Note that the sum of all weights must be equal to 1.0, if this is not the case the user is warned and the distribution is normalized automatically. Note that large particles are inserted first, so that a higher volume fraction can be achieved. If not all desired insertions could be performed, it is likely that the distribution is not accurately reproduced.

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

Information about the random state in this fix is written to binary restart files so you can restart a simulation with the same particles being chosen for insertion. None of the fix_modify options are relevant to this fix. No global scalar or vector or per-atom quantities are stored by this fix for access by various output commands. No parameter of this fix can be used with the *start*/*stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions: none

Related commands:

fix insert/stream, fix insert/pack, fix insert/rate/region

Default: none

fix particletemplate/multisphere command

Syntax:

```
fix ID group-ID particletemplate/multisphere seed keyword values nspheres nspheresvalue  ntry ntr
```

- ID, group-ID are documented in [fix](#) command
- particletemplate/multisphere = style name of this fix command
- seed = random number generator seed (integer value)
- zero or more keyword/value pairs can be appended
- keyword, values are documented in [fix particletemplate/sphere](#) command
- nspheres = obligatory keyword
- nspheresvalue = number of spheres in the template (integer value)
- ntry = obligatory keyword
- ntryvalue: number of tries for Monte Carlo approach
- spheres = obligatory keyword
- values_spheres = one out of the following options

```
    option 1 = file filename
    option 2 = file filename scale scalefactor
    option 3 = x1 y1 z1 r1 x2 y2 r2... where x/y/z are sphere positions and r are the radi
```

- type = obligatory keyword
- mt = multisphere type of the template
- opt_keyword = *mass* or *inertia_tensor* or *use_volume* or *use_density* or *fflag* or *tflag*

```
  mass value = mass assigned to this particle template
  inertia_tensor values = Ixx Ixy Ixz Iyy Iyz Izz
    Ixx Ixy Ixz Iyy Iyz Izz = 6 independant components of the inertia tensor
  use_volume = particle density calculated from mass and volume (only if keyword 'mass' is u
  use_density = particle volume calculated from mass and density (only if keyword 'mass' is
  fflag values = fflagx fflagy fflagz
    fflagx = on or off
    fflagy = on or off
    fflagz = on or off
  tflag values = tflagx tflagy tflagz
    tflagx = on or off
    tflagy = on or off
    tflagz = on or off
```

Examples:

```
fix pts1 all particletemplate/multisphere 1 atom_type 1 density constant 2500 nspheres 50 ntry 10
fix pts2 all particletemplate/multisphere 1 atom_type 1 density constant 2500 nspheres 50 ntry 10
```

Description:

Define a multisphere particle template that is used as input for a [fix_particledistribution_discrete](#) command.
The particle parameters (positions, radii) are either defined manually as arguments to this command or via a
text file, via the keyword *file*. The format of this text file is

```
x1 y1 z1 r1
x2 y2 z2 r2
...
```

and the number of lines in this file has to be equal to n_spheres as defined in this command. Comments can be made in this file via the '#' character. Optionally, when a file is used for defining the multi-sphere template, keyword *scale* can be used to define a *scalefactor* to scale up or down the particle positions and radii.

After the spheres are read, a Monte Carlo procedure is used to assess everything that is needed for the motion integration: mass, center of mass, and the inertia tensor including its eigensystem.

As an alternative, the body's mass and inertia tensor can be specified directly via keywords *mass* and *inertia_tensor*. Note that you can use these keywords only together, i.e. defining only *mass* but not *inertia_tensor* will throw an error. Also note that only 2 out of the 3 variables density, mass and volume are independant. Thus, you are offered two options when *mass* and *inertia_tensor* are used: (a) if keyword *use_volume* is specified, LIGGGHTS(R)-WSU will use the specified *mass* and volume_mc (the volume of the particle template calculated by the Monte Carlo procedure), and calculate the density from these two variables. (b) if keyword *use_density* is used, LIGGGHTS(R)-WSU will use the specified *mass* and the specified *density* (see doc of [fix particletemplate/sphere](#) command), and the volume of the clump is then calculated from these two variables. Note you have to use either *use_volume* or *use_density* in case *mass* and *inertia_tensor* are used.

The multisphere type or shape type as defined via the *type* keyword must be unique integer given to each fix particletemplate/multisphere command by the user (starting with 1), the list of all multisphere types in the simulation must be consecutive. At the moment, the multisphere type is not used, but will be used to implement orientation-dependant drag for CFD-DEM simulations in the future.

The additional keywords *fflag* and *tflag* can be used to deactivate selected translational and rotational degrees of freedom of the bodies. For example *fflag* = on on off and *tflag* = off on on would mean that bodies will not move translationally in z-direction and will not rotate around the x-axis.

IMPORTANT NOTE: As opposed to the number-based distributions, this fix uses the more common distribution based on mass-% for the radius distribution (as does [fix_particledistribution_discrete](#)).

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

Information about the random state in this fix is written to [binary restart files](#) so you can restart a simulation with the same particles being chosen for insertion. None of the [fix_modify](#) options are relevant to this fix. No global scalar or vector or per-atom quantities are stored by this fix for access by various [output commands](#). No parameter of this fix can be used with the *start/stop* keywords of the [run](#) command. This fix is not invoked during [energy minimization](#).

Restrictions: none

Related commands:

[fix_particletemplate_sphere](#)

Default:

radius = 1.0, density = 1.0, atom_type = 1, fflag = tflag = on on on

fix particletemplate/sphere command

Syntax:

```
fix ID group-ID particletemplate/sphere seed keyword values ...
```

- ID, group-ID are documented in fix command
- particletemplate/sphere = style name of this fix command
- seed = random number generator seed (integer value)
- zero or more keyword/value pairs can be appended
- keyword = *atom_type* or *density* or *volume_limit* or *radius* or *relative*

```
  atom_type value = atom type assigned to this particle template
  density values = random_style param1 (param2)
    random_style = 'constant' or 'uniform' or 'gaussian'
    param1 = density for 'constant', low value of density for 'uniform', expectancy value fo
      param2 = omitted for 'constant', high value of density for 'uniform', sigma value for
  volume_limit value = lowest particle volume allowed in simulation
  radius values = random_style param1 (param2)
    random_style = 'constant' or 'uniform number' or 'uniform mass' or 'gaussian number'
    param1 = radius for 'constant', low value of radius for 'uniform', mu value for 'gaussia
    param2 = omitted for 'constant', high value of radius for 'uniform', sigma value for 'ga
  relative value = yes or no
    no = set a flag for other commands that radius given here is absolute (in length units)
    yes = set a flag for other commands that radius given here is relative (in % of a refere
```

Examples:

```
fix pts1 all particletemplate/sphere 1 atom_type 1 density constant 2500 radius constant 0.0015
```

Description:

Define a particle that is used as input for a fix_particledistribution_discrete command. You can choose the atom type, density and radius of the particles. For density and radius, you can choose between 'constant', 'uniform' and 'gaussian' random styles. Note that for radius, you can additionally choose between a number-based and mass-based uniform distribution, where the latter is used more frequently typically. 'gaussian' for radius only supports a number-based distribution.

Keyword *relative* lets the user set a flag to let other commands know if the radii specified by this command are absolute (length units) or relative (in % of a reference radius). IMPORTANT NOTE: This setting should not be changed away from the default value (no) unless explicity required by another command!

It is thus possible to define a uniform or gaussian distribution on top of the discrete distribution defined by fix_particledistribution_discrete.

IMPORTANT NOTE: As opposed to the number-based distributions, this fix uses the more common distribution based on mass-% for the radius distribution (as does fix_particledistribution_discrete).

LIGGGHTS(R)-WSU will throw an error if the particle volume is too small compared to machine precision. If you are sure you know what you are doing you can override the default limit of 1e-12.

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

Information about the random state in this fix is written to **binary restart files** so you can restart a simulation with the same particles being chosen for insertion. None of the **fix_modify** options are relevant to this fix. No global scalar or vector or per-atom quantities are stored by this fix for access by various **output commands**. No parameter of this fix can be used with the *start*/*stop* keywords of the **run** command. This fix is not invoked during **energy minimization**.

Restrictions: none

Related commands:

fix_particletemplate_sphere

Default:

radius = 1.0, density = 1.0, atom_type = 1, volume_limit = 1e-12, relative = no

fix planeforce command

Syntax:

```
fix ID group-ID planeforce x y z
```

- ID, group-ID are documented in [fix](#) command
- lineforce = style name of this fix command
- x y z = 3-vector that is normal to the plane

Examples:

```
fix hold boundary planeforce 1.0 0.0 0.0
```

Description:

Adjust the forces on each atom in the group so that only the components of force in the plane specified by the normal vector (x,y,z) remain. This is done by subtracting out the component of force perpendicular to the plane.

If the initial velocity of the atom is 0.0 (or in the plane), then it should continue to move in the plane thereafter.

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

No information about this fix is written to [binary restart files](#). None of the [fix_modify](#) options are relevant to this fix. No global or per-atom quantities are stored by this fix for access by various [output commands](#). No parameter of this fix can be used with the *start/stop* keywords of the [run](#) command.

The forces due to this fix are imposed during an energy minimization, invoked by the [minimize](#) command.

Restrictions: none

Related commands:

[fix lineforce](#)

Default: none

fix poems command

Syntax:

```
fix ID group-ID poems keyword values
```

- ID, group-ID are documented in [fix](#) command
- poems = style name of this fix command
- keyword = *group* or *file* or *molecule*

```
  group values = list of group IDs
    molecule values = none
      file values = filename
```

Examples:

```
fix 3 fluid poems group clump1 clump2 clump3
fix 3 fluid poems file cluster.list
```

Description:

Treats one or more sets of atoms as coupled rigid bodies. This means that each timestep the total force and torque on each rigid body is computed and the coordinates and velocities of the atoms are updated so that the collection of bodies move as a coupled set. This can be useful for treating a large biomolecule as a collection of connected, coarse-grained particles.

The coupling, associated motion constraints, and time integration is performed by the software package [Parallelizable Open source Efficient Multibody Software (POEMS)](#) which computes the constrained rigid-body motion of articulated (jointed) multibody systems [(Anderson)](#). POEMS was written and is distributed by Prof Kurt Anderson, his graduate student Rudranarayan Mukherjee, and other members of his group at Rensselaer Polytechnic Institute (RPI). Rudranarayan developed the LIGGGHTS(R)-WSU/POEMS interface. For copyright information on POEMS and other details, please refer to the documents in the poems directory distributed with LIGGGHTS(R)-WSU.

This fix updates the positions and velocities of the rigid atoms with a constant-energy time integration, so you should not update the same atoms via other fixes (e.g. nve, nvt, npt, temp/rescale, langevin).

Each body must have a non-degenerate inertia tensor, which means if must contain at least 3 non-collinear atoms. Which atoms are in which bodies can be defined via several options.

For option *group*, each of the listed groups is treated as a rigid body. Note that only atoms that are also in the fix group are included in each rigid body.

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

For option *file*, sets of atoms are read from the specified file and each set is treated as a rigid body. Each line of the file specifies a rigid body in the following format:

ID type atom1-ID atom2-ID atom3-ID ...

ID as an integer from 1 to M (the number of rigid bodies). Type is any integer; it is not used by the fix poems command. The remaining arguments are IDs of atoms in the rigid body, each typically from 1 to N (the

number of atoms in the system). Only atoms that are also in the fix group are included in each rigid body. Blank lines and lines that begin with '#' are skipped.

A connection between a pair of rigid bodies is inferred if one atom is common to both bodies. The POEMS solver treats that atom as a spherical joint with 3 degrees of freedom. Currently, a collection of bodies can only be connected by joints as a linear chain. The entire collection of rigid bodies can represent one or more chains. Other connection topologies (tree, ring) are not allowed, but will be added later. Note that if no joints exist, it is more efficient to use the fix rigid command to simulate the system.

When the poems fix is defined, it will print out statistics on the total # of clusters, bodies, joints, atoms involved. A cluster in this context means a set of rigid bodies connected by joints.

For computational efficiency, you should turn off pairwise and bond interactions within each rigid body, as they no longer contribute to the motion. The "neigh_modify exclude" and "delete_bonds" commands can be used to do this if each rigid body is a group.

For computational efficiency, you should only define one fix poems which includes all the desired rigid bodies. LIGGGHTS(R)-WSU will allow multiple poems fixes to be defined, but it is more expensive.

The degrees-of-freedom removed by coupled rigid bodies are accounted for in temperature and pressure computations. Similarly, the rigid body contribution to the pressure virial is also accounted for. The latter is only correct if forces within the bodies have been turned off, and there is only a single fix poems defined.

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

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix. No global or per-atom quantities are stored by this fix for access by various output commands. No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions:

This fix is part of the POEMS package. It is only enabled if LIGGGHTS(R)-WSU was built with that package, which also requires the POEMS library be built and linked with LIGGGHTS(R)-WSU. See the Making LIGGGHTS(R)-WSU section for more info.

Related commands:

fix rigid, delete_bonds, neigh_modify exclude

Default: none

(Anderson) Anderson, Mukherjee, Critchley, Ziegler, and Lipton "POEMS: Parallelizable Open-source Efficient Multibody Software ", Engineering With Computers (2006). (link to paper)

fix print command

Syntax:

```
fix ID group-ID print N string keyword value ...
```

- ID, group-ID are documented in fix command
- print = style name of this fix command
- N = print every N steps
- string = text string to print with optional variable names
- zero or more keyword/value pairs may be appended
- keyword = *file* or *append* or *screen* or *title*

```
  file value = filename
    append value = filename
    screen value = yes or no
    title value = string
      string =  text to print as 1st line of output file
```

Examples:

```
fix extra all print 100 "Coords of marker atom = $x $y $z"
fix extra all print 100 "Coords of marker atom = $x $y $z" file coord.txt
```

Description:

Print a text string every N steps during a simulation run. This can be used for diagnostic purposes or as a debugging tool to monitor some quantity during a run. The text string must be a single argument, so it should be enclosed in double quotes if it is more than one word. If it contains variables it must be enclosed in double quotes to insure they are not evaluated when the input script line is read, but will instead be evaluated each time the string is printed.

See the variable command for a description of *equal* style variables which are the most useful ones to use with the fix print command, since they are evaluated afresh each timestep that the fix print line is output. Equal-style variables calculate formulas involving mathematical operations, atom properties, group properties, thermodynamic properties, global values calculated by a compute or fix, or references to other variables.

If the *file* or *append* keyword is used, a filename is specified to which the output generated by this fix will be written. If *file* is used, then the filename is overwritten if it already exists. If *append* is used, then the filename is appended to if it already exists, or created if it does not exist.

If the *screen* keyword is used, output by this fix to the screen and logfile can be turned on or off as desired.

The *title* keyword allow specification of the string that will be printed as the first line of the output file, assuming the *file* keyword was used. By default, the title line is as follows:

```
# Fix print output for fix ID
```

where ID is replaced with the fix-ID. if *title* = 'none', then no title line will be printed

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

No information about this fix is written to **binary restart files**. None of the **fix_modify** options are relevant to this fix. No global or per-atom quantities are stored by this fix for access by various **output commands**. No parameter of this fix can be used with the *start/stop* keywords of the **run** command. This fix is not invoked during **energy minimization**.

Restrictions: none

Related commands:

variable, print

Default:

The option defaults are no file output, screen = yes, and title string as described above.

fix property/atom/tracer command

Syntax:

```
fix id group property/atom/tracer region_mark region-ID mark_step s keyword value ...
```

- ID, group-ID are documented in [fix](#) command
- property/atom/tracer = style name of this fix command
- region_mark = obligatory keyword
- region-ID = ID of region atoms must be in to be marked
- mark_step = obligatory keyword
- s = step when atoms are marked (or started to be marked, depending on marker_style)
- zero or more keyword/value pairs may be appended to args
- keyword = *marker_style* or *check_mark_every*

```
  marker_style value = dirac or heaviside
      dirac = use a dirac impulse at time step s to mark the particles
      heaviside = use a dirac impulse staring at time step s to mark the particles
  check_mark_every value = n
      n = check every that many time-step if atom are in region to be marked
```

Examples:

```
fix tracer all property/atom/tracer region_mark mark mark_step 10000 marker_style dirac check_mar
```

Description:

Fix property/atom/tracer marks particles using either a Dirac delta impulse (default) or a Heaviside impulse, as specified by the *marker_style* keyword. Particles are marked if they are inside the region specified by the *region_mark* keyword. Using the Dirac impulse means that all the particles which are in the region at the time-step specified by the *mark_step* keyword are marked. Using the Heaviside impulse means that all the particles which pass by the specified region after the specified time-step are marked.

Keyword *check_mark_every* can be used to control how often the region is checked. Typically, this is useful when the *heaviside* option is used, because you may not want to check each particle each time-step. However, be careful not to choose this value too large, in this case you could skip particles passing through the region.

It is useful to combine this command with a [compute nparticles/tracer/region](#) command to compute residence time distributions.

IMPORTANT NOTE: Using [compute nparticles/tracer/region](#) can change the tracer value (keyword **reset_marker**.)

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

Information about this fix is written to [binary restart files](#) .

This fix computes a per-atom vector (the marker) which can be accessed by various [output commands](#). . This fix also computes a global scalar indicating how many particles were marked since the last time the global scalar was computed. This scalar can also be accessed by various [output commands](#). .

Restrictions:

Currently, this feature does not support multi-sphere particles.

Related commands:

compute nparticles/tracer/region

Default:

marker_style = dirac, check_mark_every = 10

fix property/atom/tracer/stream command

Syntax:

```
fix id group property/atom/tracer/stream mark_step s n_tracer n insert_stream ins-ID every e
```

- ID, group-ID are documented in fix command
- property/atom/tracer/stream = style name of this fix command
- mark_step = obligatory keyword
- s = step when atoms are marked (or started to be marked, depending on marker_style)
- n_tracer = obligatory keyword
- n = number of tracer atoms to be marked
- insert_stream = obligatory keyword
- ins-ID = ID of a fix insert/stream
- every = obligatory keyword

```
  e = 'once' or integer > 0
```

Examples:

```
fix tracer all property/atom/tracer/stream mark_step 10000  insert_stream ins  n_tracer 20  every
```

Description:

Fix property/atom/tracer/stream marks a given number of particles (as defined by keyword *n_tracer*) which are generated by a fix insert/stream command (as defined by keyword *ins-ID*. The first *n_tracer* particles which pass the insertion face after time-step *mark_step* are being marked as tracers. In case of **every** = once, this procedure is performed once, otherwise the procedure is repeated for the first *n_tracer* particles which pass the insertion face after step *mark_step* + *every*.

Note that even for option *once*, particles are the marking procedure can extend over multiple packets of insertion by a fix insert/stream in case that the number of particles inserted in a packet is smaller than the number of particles to tag (as defined by keyword *n_tracer*).

An arbitrary number of fix property/atom/tracer/stream commands can be used for a given fix insert/stream.

It is useful to combine this command with a compute nparticles/tracer/region command to compute residence time distributions.

IMPORTANT NOTE: Due to some parallel operation which needed to tag the particles, you need an atom map to be allocated, see the atom_modify command for details.

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

Information about this fix is written to binary restart files .

This fix computes a per-atom vector (the marker) which can be accessed by various output commands. . This fix also computes a global scalar indicating how many particles were marked since the last time the global scalar was computed. This scalar can also be accessed by various output commands. .

Restrictions:

Currently, this feature does not support multi-sphere particles.

Related commands:

compute nparticles/tracer/region

Default: none

fix property/global command

fix property/atom command

Syntax:

```
fix id group property/atom variablename style restartvalue comm_ghost_value comm_reverse_ghost_va
fix id group property/global variablename style stylearg defaultvalue(s)...
```

- ID, group-ID are documented in [fix](#) command
- property/global or property/atom = style name of this fix command
- variablename = a valid C++ string
- restartvalues = 'yes' or 'no'
- comm_ghost_value = 'yes' or 'no'
- comm_reverse_ghost_value = 'yes' or 'no'

fix property/global:

- style = scalar or vector or atomtype or matrix or atomtypepair

```
        stylearg for scalar/vector: none
        stylearg for matrix/atomtypepair: nCols
```

fix property/atom:

- style = scalar or vector
- restartvalue = yes or no
- communicate_ghost_value = yes or no
- communicate_reverse_ghost_value = yes or no

Examples:

```
fix m3 all property/global coefficientRestitution peratomtypepair 1 0.3
fix m5 all property/global characteristicVelocity scalar 2.
fix uf all property/atom uf vector yes no no  0. 0. 0.
```

Description:

Fix property/atom reserves per-atom properties to be accessed by the user or other fixes. Style *scalar* reserves one value per atom, style *vector* multiple values per atoms, where the number of *defaultvalues* (that are assigned to the atoms at creation) determines the length of the vector. The group of atoms the fix is applied to is always "all", irrespective of which group is used for the fix command . If you want to assign different values for different groups, you can use the [set](#) command with keyword 'property/atom'. Keyword *restartvalues* determines whether information about the values stored by this fix is written to binary restart files. Keyword *communicate_ghost_value* determines whether information about the values stored by this fix can be communicated to ghost particles (forward communication). The exact location during a time-step when this happens depends on the model that uses this fix. Keyword *communicate_reverse_ghost_value* determines whether information about the values stored by this fix can be communicated from ghost particles to owned particles (reverse communication). The exact location during a time-step when this happens depends on the model that uses this fix.

Fix property/global reserves global properties to be accessed by the user or other fixes or pair styles. The number of defaultvalues determines the length of the vector / the number of matrix components . For style *vector* or *atomtype*, the user provides the number of vector components . For style *matrix* or *atomtypepair*, the user provides the number of matrix columns (*nCols*) .

Example: *nCols*= 2 and *defaultvalues* = 1 2 3 4 5 6 would be mapped into a matrix like

[1 2]

[3 4]

[5 6]

Note that the number of default values must thus be a multiple of *nCols*. Note that *vector* and *atomtype* do the same thing, *atomtype* is just provided to make input scripts more readable . Note that *matrix* and *atomtypepair* both refer to a matrix of global values. However, a matrix defined via *atomtypepair* is required to be symmetric.

Note that the group of atoms the fix is applied to is ignored (as the fix is not applied to atoms, but defines values of global scope).

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

Information about this fix is written to **binary restart files** if you set *restartvalue* to 'yes'.

Restrictions: none

Related commands:

set, pair_gran

Default: none

fix rigid command

fix rigid/nve command

fix rigid/nvt command

fix rigid/npt command

fix rigid/nph command

fix rigid/small command

Syntax:

```
fix ID group-ID style bodystyle args keyword values ...
```

- ID, group-ID are documented in [fix](#) command
- style = *rigid*
- bodystyle = *single* or *molecule* or *group*

```
  single args = none
    molecule args = none
    group args = N groupID1 groupID2 ...
      N = # of groups
      groupID1, groupID2, ... = list of N group IDs
```

- zero or more keyword/value pairs may be appended
- keyword = or *temp* or *iso* or *aniso* or *x* or *y* or *z* or *couple* or *tparam* or *pchain* or *dilate* or *force* or *torque* or *infile*

```
  langevin values = Tstart Tstop Tperiod seed
      Tstart,Tstop = desired temperature at start/stop of run (temperature units)
      Tdamp = temperature damping parameter (time units)
      seed = random number seed to use for white noise (positive integer)
    temp values = Tstart Tstop Tdamp
      Tstart,Tstop = desired temperature at start/stop of run (temperature units)
      Tdamp = temperature damping parameter (time units)
    iso or aniso values = Pstart Pstop Pdamp
      Pstart,Pstop = scalar external pressure at start/end of run (pressure units)
      Pdamp = pressure damping parameter (time units)
    x or y or z values = Pstart Pstop Pdamp
      Pstart,Pstop = external stress tensor component at start/end of run (pressure units)
      Pdamp = stress damping parameter (time units)
    couple = none or xyz or xy or yz or xz
    tparam values = Tchain Titer Torder
      Tchain = length of Nose/Hoover thermostat chain
      Titer = number of thermostat iterations performed
      Torder = 3 or 5 = Yoshida-Suzuki integration parameters
    pchain values = Pchain
      Pchain = length of the Nose/Hoover thermostat chain coupled with the barostat
    dilate value = dilate-group-ID
      dilate-group-ID = only dilate atoms in this group due to barostat volume changes
    force values = M xflag yflag zflag
      M = which rigid body from 1-Nbody (see asterisk form below)
      xflag,yflag,zflag = off/on if component of center-of-mass force is active
    torque values = M xflag yflag zflag
      M = which rigid body from 1-Nbody (see asterisk form below)
```

```
        xflag,yflag,zflag = off/on if component of center-of-mass torque is active
    infile filename
        filename = file with per-body values of mass, center-of-mass, moments of inertia
```

Examples:

```
fix 1 clump rigid single
fix 2 fluid rigid group 3 clump1 clump2 clump3 torque * off off off
```

Description:

Treat one or more sets of atoms as independent rigid bodies. This means that each timestep the total force and torque on each rigid body is computed as the sum of the forces and torques on its constituent particles and the coordinates, velocities, and orientations of the atoms in each body are updated so that the body moves and rotates as a single entity.

Examples of large rigid bodies are a large colloidal particle, or portions of a large biomolecule such as a protein.

Example of small rigid bodies are patchy nanoparticles, such as those modeled in this paper by Sharon Glotzer's group, clumps of granular particles, lipid molecules consiting of one or more point dipoles connected to other spheroids or ellipsoids, irregular particles built from line segments (2d) or triangles (3d), and coarse-grain models of nano or colloidal particles consisting of a small number of constituent particles. Note that the fix shake command can also be used to rigidify small molecules of 2, 3, or 4 atoms, e.g. water molecules. That fix treats the constituent atoms as point masses.

These fixes also update the positions and velocities of the atoms in each rigid body via time integration, in the NVE ensemble.

IMPORTANT NOTE: Not all of the bodystyle options and keyword/value options are available for both the *rigid* and *rigid/small* variants. See details below.

The *rigid* variant is typically the best choice for a system with a small number of large rigid bodies, each of which can extend across the domain of many processors. It operates by creating a single global list of rigid bodies, which all processors contribute to. MPI_Allreduce operations are performed each timestep to sum the contributions from each processor to the force and torque on all the bodies. This operation will not scale well in parallel if large numbers of rigid bodies are simulated.

Which of the two variants is faster for a particular problem is hard to predict. The best way to decide is to perform a short test run. Both variants should give identical numerical answers for short runs. Long runs should give statistically similar results, but round-off differences will accumulate to produce divergent trajectories.

IMPORTANT NOTE: You should not update the atoms in rigid bodies via other time-integration fixes (e.g. fix nve), or you will be integrating their motion more than once each timestep. When performing a hybrid simulation with some atoms in rigid bodies, and some not, a separate time integration fix like fix nve should be used for the non-rigid particles.

IMPORTANT NOTE: These fixes are overkill if you simply want to hold a collection of atoms stationary or have them move with a constant velocity. A simpler way to hold atoms stationary is to not include those atoms in your time integration fix. E.g. use "fix 1 mobile nve" instead of "fix 1 all nve", where "mobile" is the group of atoms that you want to move. You can move atoms with a constant velocity by assigning them an initial velocity (via the velocity command), setting the force on them to 0.0 (via the fix setforce command), and integrating them as usual (e.g. via the fix nve command).

Each rigid body must have two or more atoms. An atom can belong to at most one rigid body. Which atoms are in which bodies can be defined via several options.

For bodystyle *single* the entire fix group of atoms is treated as one rigid body. This option is only allowed for fix rigid and its sub-styles.

For bodystyle *molecule*, each set of atoms in the fix group with a different molecule ID is treated as a rigid body. This option is allowed for fix rigid and fix rigid/small, and their sub-styles. Note that atoms with a molecule ID = 0 will be treated as a single rigid body. For a system with atomic solvent (typically this is atoms with molecule ID = 0) surrounding rigid bodies, this may not be what you want. Thus you should be careful to use a fix group that only includes atoms you want to be part of rigid bodies.

For bodystyle *group*, each of the listed groups is treated as a separate rigid body. Only atoms that are also in the fix group are included in each rigid body. This option is only allowed for fix rigid and its sub-styles.

IMPORTANT NOTE: To compute the initial center-of-mass position and other properties of each rigid body, the image flags for each atom in the body are used to "unwrap" the atom coordinates. Thus you must insure that these image flags are consistent so that the unwrapping creates a valid rigid body (one where the atoms are close together), particularly if the atoms in a single rigid body straddle a periodic boundary. This means the input data file or restart file must define the image flags for each atom consistently or that you have used the set command to specify them correctly. If a dimension is non-periodic then the image flag of each atom must be 0 in that dimension, else an error is generated.

The *force* and *torque* keywords discussed next are only allowed for fix rigid and its sub-styles.

By default, each rigid body is acted on by other atoms which induce an external force and torque on its center of mass, causing it to translate and rotate. Components of the external center-of-mass force and torque can be turned off by the *force* and *torque* keywords. This may be useful if you wish a body to rotate but not translate, or vice versa, or if you wish it to rotate or translate continuously unaffected by interactions with other particles. Note that if you expect a rigid body not to move or rotate by using these keywords, you must insure its initial center-of-mass translational or angular velocity is 0.0. Otherwise the initial translational or angular momentum the body has will persist.

An xflag, yflag, or zflag set to *off* means turn off the component of force of torque in that dimension. A setting of *on* means turn on the component, which is the default. Which rigid body(s) the settings apply to is determined by the first argument of the *force* and *torque* keywords. It can be an integer M from 1 to Nbody, where Nbody is the number of rigid bodies defined. A wild-card asterisk can be used in place of, or in conjunction with, the M argument to set the flags for multiple rigid bodies. This takes the form "*" or "*n" or "n*" or "m*n". If N = the number of rigid bodies, then an asterisk with no numeric values means all bodies from 1 to N. A leading asterisk means all bodies from 1 to n (inclusive). A trailing asterisk means all bodies from n to N (inclusive). A middle asterisk means all types from m to n (inclusive). Note that you can use the *force* or *torque* keywords as many times as you like. If a particular rigid body has its component flags set multiple times, the settings from the final keyword are used.

For computational efficiency, you may wish to turn off pairwise and bond interactions within each rigid body, as they no longer contribute to the motion. The neigh_modify exclude and delete_bonds commands are used to do this.

For computational efficiency, you should typically define one fix rigid or fix rigid/small command which includes all the desired rigid bodies. LIGGGHTS(R)-WSU will allow multiple rigid fixes to be defined, but it is more expensive.

The constituent particles within a rigid body can be point particles (the default in LIGGGHTS(R)-WSU) or finite-size particles, such as spheres or ellipsoids or line segments or triangles. See the atom_style sphere and

fix rigid/small command 279

[ellipsoid and line and tri](#) commands for more details on these kinds of particles. Finite-size particles contribute differently to the moment of inertia of a rigid body than do point particles. Finite-size particles can also experience torque (e.g. due to [frictional granular interactions](#)) and have an orientation. These contributions are accounted for by these fixes.

Forces between particles within a body do not contribute to the external force or torque on the body. Thus for computational efficiency, you may wish to turn off pairwise and bond interactions between particles within each rigid body. The [neigh_modify exclude](#) and [delete_bonds](#) commands are used to do this. For finite-size particles this also means the particles can be highly overlapped when creating the rigid body.

The *rigid* style performs constant NVE time integration based on Richardson iterations.

The *infile* keyword allows a file of rigid body attributes to be read in from a file, rather then having LIGGGHTS(R)-WSU compute them. There are 3 such attributes: the total mass of the rigid body, its center-of-mass position, and its 6 moments of inertia. For rigid bodies consisting of point particles or non-overlapping finite-size particles, LIGGGHTS(R)-WSU can compute these values accurately. However, for rigid bodies consisting of finite-size particles which overlap each other, LIGGGHTS(R)-WSU will ignore the overlaps when computing these 3 attributes. The amount of error this induces depends on the amount of overlap. To avoid this issue, the values can be pre-computed (e.g. using Monte Carlo integration).

The format of the file is as follows. Note that the file does not have to list attributes for every rigid body integrated by fix rigid. Only bodies which the file specifies will have their computed attributes overridden. The file can contain initial blank lines or comment lines starting with "#" which are ignored. The first non-blank, non-comment line should list N = the number of lines to follow. The N successive lines contain the following information:

```
ID1 masstotal xcm ycm zcm ixx iyy izz ixy ixz iyz
ID2 masstotal xcm ycm zcm ixx iyy izz ixy ixz iyz
...
IDN masstotal xcm ycm zcm ixx iyy izz ixy ixz iyz
```

The rigid body IDs are all positive integers. For the *single* bodystyle, only an ID of 1 can be used. For the *group* bodystyle, IDs from 1 to Ng can be used where Ng is the number of specified groups. For the *molecule* bodystyle, use the molecule ID for the atoms in a specific rigid body as the rigid body ID.

The masstotal and center-of-mass coordinates (xcm,ycm,zcm) are self-explanatory. The center-of-mass should be consistent with what is calculated for the position of the rigid body with all its atoms unwrapped by their respective image flags. If this produces a center-of-mass that is outside the simulation box, LIGGGHTS(R)-WSU wraps it back into the box. The 6 moments of inertia (ixx,iyy,izz,ixy,ixz,iyz) should be the values consistent with the current orientation of the rigid body around its center of mass. The values are with respect to the simulation box XYZ axes, not with respect to the prinicpal axes of the rigid body itself. LIGGGHTS(R)-WSU performs the latter calculation internally.

IMPORTANT NOTE: If you use the *infile* keyword and write restart files during a simulation, then each time a restart file is written, the fix also write an auxiliary restart file with the name rfile.rigid, where "rfile" is the name of the restart file, e.g. tmp.restart.10000 and tmp.restart.10000.rigid. This auxiliary file is in the same format described above and contains info on the current center-of-mass and 6 moments of inertia. Thus it can be used in a new input script that restarts the run and re-specifies a rigid fix using an *infile* keyword and the appropriate filename. Note that the auxiliary file will contain one line for every rigid body, even if the original file only listed a subset of the rigid bodies.

IMPORTANT NOTE: The periodic image flags of atoms in rigid bodies are altered so that the rigid body can be reconstructed correctly when it straddles periodic boundaries. The atom image flags are not incremented/decremented as they would be for non-rigid atoms as the rigid body crosses periodic boundaries. Specifically, they are set so that the center-of-mass (COM) of the rigid body always remains inside the

simulation box.

This means that if you output per-atom image flags you cannot interpret them as you normally would. I.e. the image flag values written to a dump file will be different than they would be if the atoms were not in a rigid body. Likewise the compute msd will not compute the expected mean-squared displacement for such atoms if the body moves across periodic boundaries. It also means that if you have bonds between a pair of rigid bodies and the bond straddles a periodic boundary, you cannot use the replicate command to increase the system size.

Here are details on how, you can post-process a dump file to calculate a diffusion coefficient for rigid bodies, using the altered per-atom image flags written to a dump file. The image flags for atoms in the same rigid body can be used to unwrap the body and calculate its center-of-mass (COM). As mentioned above, this COM will always be inside the simulation box. Thus it will "jump" from one side of the box to the other when the COM crosses a periodic boundary. If you keep track of the jumps, you can effectively "unwrap" the COM and use that value to track the displacement of each rigid body, and thus the mean-squared displacement (MSD) of an ensemble of bodies, and thus a diffusion coefficient.

Note that fix rigid does define image flags for each rigid body, which are incremented when the center-of-mass of the rigid body crosses a periodic boundary in the usual way. These image flags have the same meaning as atom images (see the "dump" command) and can be accessed and output as described below.

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

No information is written to binary restart files. read_restart command for info on how to re-specify a fix in an input script that reads a restart file, so that the operation of the fix continues in an uninterrupted fashion.

The fix_modify *energy* option is supported by the rigid/nvt fix to add the energy change induced by the thermostatting to the system's potential energy as part of thermodynamic output.

The fix_modify *temp* and *press* options are supported by the rigid/npt and rigid/nph fixes to change the computes used to calculate the instantaneous pressure tensor. Note that the rigid/nvt fix does not use any external compute to compute instantaneous temperature.

The fixes compute a global scalar which can be accessed by various output commands. The scalar value calculated by these fixes is "intensive". The scalar is the current temperature of the collection of rigid bodies. This is averaged over all rigid bodies and their translational and rotational degrees of freedom. The translational energy of a rigid body is 1/2 m v^2, where m = total mass of the body and v = the velocity of its center of mass. The rotational energy of a rigid body is 1/2 I w^2, where I = the moment of inertia tensor of the body and w = its angular velocity. Degrees of freedom constrained by the *force* and *torque* keywords are removed from this calculation, but only for the *rigid* and *rigid/nve* fixes.

This fix computes a global array of values which can be accessed by various output commands. The number of rows in the array is equal to the number of rigid bodies. The number of columns is 15. Thus for each rigid body, 15 values are stored: the xyz coords of the center of mass (COM), the xyz components of the COM velocity, the xyz components of the force acting on the COM, the xyz components of the torque acting on the COM, and the xyz image flags of the COM, which have the same meaning as image flags for atom positions (see the "dump" command). The force and torque values in the array are not affected by the *force* and *torque* keywords in the fix rigid command; they reflect values before any changes are made by those keywords.

The ordering of the rigid bodies (by row in the array) is as follows. For the *single* keyword there is just one rigid body. For the *molecule* keyword, the bodies are ordered by ascending molecule ID. For the *group* keyword, the list of group IDs determines the ordering of bodies.

The array values calculated by these fixes are "intensive", meaning they are independent of the number of

fix rigid/small command   281

atoms in the simulation.

No parameter of these fixes can be used with the *start/stop* keywords of the run command. These fixes are not invoked during energy minimization.

Restrictions:

These fixes are all part of the RIGID package. It is only enabled if LIGGGHTS(R)-WSU was built with that package. See the Making LIGGGHTS(R)-WSU section for more info.

Related commands:

delete_bonds, neigh_modify exclude

Default:

The option defaults are force * on on on and torque * on on on, meaning all rigid bodies are acted on by center-of-mass force and torque. Also Tchain = Pchain = 10, Titer = 1, Torder = 3.

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

(Kamberaj) Kamberaj, Low, Neal, J Chem Phys, 122, 224114 (2005).

(Martyna) Martyna, Klein, Tuckerman, J Chem Phys, 97, 2635 (1992); Martyna, Tuckerman, Tobias, Klein, Mol Phys, 87, 1117.

(Miller) Miller, Eleftheriou, Pattnaik, Ndirango, and Newns, J Chem Phys, 116, 8649 (2002).

(Zhang) Zhang, Glotzer, Nanoletters, 4, 1407-1413 (2004).

fix setforce command

Syntax:

```
fix ID group-ID setforce fx fy fz keyword value ...
```

- ID, group-ID are documented in [fix](#) command
- setforce = style name of this fix command
- fx,fy,fz = force component values
- any of fx,fy,fz can be a variable (see below)
- zero or more keyword/value pairs may be appended to args
- keyword = *region*

```
  region value = region-ID
      region-ID = ID of region atoms must be in to have added force
```

Examples:

```
fix freeze indenter setforce 0.0 0.0 0.0
fix 2 edge setforce NULL 0.0 0.0
fix 2 edge setforce NULL 0.0 v_oscillate
```

Description:

Set each component of force on each atom in the group to the specified values fx,fy,fz. This erases all previously computed forces on the atom, though additional fixes could add new forces. This command can be used to freeze certain atoms in the simulation by zeroing their force, either for running dynamics or performing an energy minimization. For dynamics, this assumes their initial velocity is also zero.

Any of the fx,fy,fz values can be specified as NULL which means do not alter the force component in that dimension.

Any of the 3 quantities defining the force components can be specified as an equal-style or atom-style [variable](#), namely *fx*, *fy*, *fz*. If the value is a variable, it should be specified as v_name, where name is the variable name. In this case, the variable will be evaluated each timestep, and its value used to determine the force component.

Equal-style variables can specify formulas with various mathematical functions, and include [thermo_style](#) command keywords for the simulation box parameters and timestep and elapsed time. Thus it is easy to specify a time-dependent force field.

Atom-style variables can specify the same formulas as equal-style variables but can also include per-atom values, such as atom coordinates. Thus it is easy to specify a spatially-dependent force field with optional time-dependence as well.

If the *region* keyword is used, the atom must also be in the specified geometric [region](#) in order to have force added to it.

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

No information about this fix is written to **binary restart files**. None of the **fix_modify** options are relevant to this fix.

This fix computes a global 3-vector of forces, which can be accessed by various **output commands**. This is the total force on the group of atoms before the forces on individual atoms are changed by the fix. The vector values calculated by this fix are "extensive".

No parameter of this fix can be used with the *start/stop* keywords of the **run** command.

The forces due to this fix are imposed during an energy minimization, invoked by the **minimize** command, but you cannot set forces to any value besides zero when performing a minimization. Use the **fix addforce** command if you want to apply a non-zero force to atoms during a minimization.

Restrictions: none

Related commands:

fix addforce, fix aveforce

Default: none

fix sph/density/continuity command

Syntax:

```
fix ID group-ID sph/density/continuity
```

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

Examples:

```
fix density all sph/density/continuity
```

Description:

Based on the continuity equation in the form

$$\frac{d\rho}{dt} = -\nabla \cdot (\rho \vec{v}) + \vec{v} \cdot \nabla \rho$$

this fix calculates the density of each particle by the rule

$$\frac{d\rho_a}{dt} = \sum_b m_b \left(\vec{v}_a - \vec{v}_b \right) \cdot \nabla_a W_{ab}$$

where the summation is over all particles b other than particle a, m is the mass, v is the velocity, W_{ab} is the interpolating kernel (documented in [pair_style sph/artVisc/tensCorr](#)) and ∇_a is the gradient of W_{ab}.

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

No information about this fix is written to [binary restart files](#).

None of the [fix_modify](#) options are relevant to this fix.

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

No parameter of this fix can be used with the *start/stop* keywords of the [run](#) command.

This fix is not invoked during [energy minimization](#).

Restrictions:

There can be only one fix sph/density.

Related commands:

[pair_style sph/artVisc/tensCorr](#), [fix sph/pressure](#), [fix sph/density/corr](#)

Default: none

(Liu and Liu, 2003) "Smoothed Particle Hydrodynamics: A Meshfree Particle Method", G. R. Liu and M. B. Liu, World Scientific, p. 449 (2003).

(Monaghan, 1992) "Smoothed Particle Hydrodynamics", J. J. Monaghan, Annu. Rev. Astron. Astrophys., 30, p. 543-574 (1992).

fix sph/density/corr command

Syntax:

```
fix ID group-ID sph/density/corr style args
```

- ID, group-ID are documented in [fix](#) command
- sph/density/corr = style name of this fix command
- style = *shepard*
- args = list of arguments for a particular style

shepard args = every nSteps
    nSteps = determes number of timesteps

Examples:

```
fix corr all sph/density/corr shepard every 30
```

Description:

In general the pressure field in SPH exhibits large oscillations. One approach to overcome this problem is to perform a filter over the density.

The filterstyle *shepard* is one of the most simple and quick correction. Every *nSteps* timesteps the following rule is applied:

$$\rho_a^{new} = \frac{\sum_b m_b W_{ab}}{\sum_b \frac{m_b}{\rho_b} W_{ab}}$$

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

No information about this fix is written to [binary restart files](#).

None of the [fix_modify](#) options are relevant to this fix.

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

No parameter of this fix can be used with the *start/stop* keywords of the [run](#) command.

This fix is not invoked during [energy minimization](#).

Restrictions: none

Related commands:

[pair_style sph/artVisc/tensCorr](#), [fix sph/pressure](#), [fix sph/density/continuity](#)

Default: none

(Liu and Liu, 2003) "Smoothed Particle Hydrodynamics: A Meshfree Particle Method", G. R. Liu and M. B. Liu, World Scientific, p. 449 (2003).

fix sph/density/summation command

Syntax:

```
fix ID group-ID sph/density/summation
```

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

Examples:

```
fix density all sph/density/summation
```

Description:

Calculates the density field with the classic SPH-summation approach. The governing equation is given by:

$$\rho_a = \sum_b m_b W_{ab}$$

rho_a is the density of particle a, m is the mass and W_{ab} denotes the interpolating kernel for the particle-particle distance r_a - r_b. The summation is over all particles b other than particle a.

NOTE: In the current version boundary or image particles are not implemented. Therefore, the density calculation in the vicinity to a wall will be wrong.

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

No information about this fix is written to [binary restart files](#).

None of the [fix_modify](#) options are relevant to this fix.

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

No parameter of this fix can be used with the *start/stop* keywords of the [run](#) command.

This fix is not invoked during [energy minimization](#).

Restrictions:

There can be only one fix sph/density/... (except [fix sph/density/corr](#))

Related commands:

[pair_style sph/artVisc/tensCorr](#), [fix sph/pressure](#), [fix sph/density/continuity](#)

Default: none

(Liu and Liu, 2003) "Smoothed Particle Hydrodynamics: A Meshfree Particle Method", G. R. Liu and M. B. Liu, World Scientific, p. 449 (2003).

(Monaghan, 1992) "Smoothed Particle Hydrodynamics", J. J. Monaghan, Annu. Rev. Astron. Astrophys., 30, p. 543-574 (1992).

fix sph/pressure command

Syntax:

```
fix ID group-ID sph/pressure style args
```

- ID, group-ID are documented in <u>fix</u> command
- sph/pressure = style name of this fix command
- style = *absolut* or *Tait*
- args = list of arguments for a particular style

```
 absolut args = NULL
    Tait args = B density0 gamma
      B = coefficient
      density0 = reference density
      gamma = isentropic exponent
```

Examples:

```
fix pressure all sph/pressure absolut
fix pressure all sph/pressure Tait 2000000. 1000. 7.
```

LIGGGHTS vs. LAMMPS Info:

This command is not available in LAMMPS.

Description:

The equation of state (EOS) for the SPH calculation is the link between the density field and the pressure field. A lot of different equations can be found in the literature.

The *absolut* style was the first implemented EOS. Based on "An initiation to SPH" from Lucas Braune and Thomas Lewiner this simple equation

$$P_a = 0.1\rho_a^2$$

is implemented, where $_a$ is the density of particle a.

In case of *Tait* style the rule

$$P_a = B\left[\left(\frac{\rho_a}{\rho_0}\right)^{\gamma} - 1\right]$$

is applied. B denotes the pressure prefactor which is calculated by

$$B = \frac{c_0^2 \rho_0}{\gamma}$$

where c_0 is the speed of sound of the material. $_0$ is the reference density and is the isentropic exponent defined as c_p/c_v.

NOTE: Monaghan has found that the speed of sound could be artificially reduced. (Monaghan, 1994) Therefore, we can choose a greater time step. He argues that the minimum sound speed should be about ten times greater than the maximum expected flow speed. (Δ < 1%)

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

No information about this fix is written to [binary restart files](#).

None of the [fix_modify](#) options are relevant to this fix.

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

No parameter of this fix can be used with the *start/stop* keywords of the [run](#) command.

This fix is not invoked during [energy minimization](#).

Restrictions:

One fix sph/density/summation (only dev-version) or sph/denstiy/continuity has to exist.

Related commands:

[pair_style sph](#), [fix sph/density/continuity](#)

Default: none

fix spring command

Syntax:

```
fix ID group-ID spring keyword values
```

- ID, group-ID are documented in [fix](#) command
- spring = style name of this fix command
- keyword = *tether* or *couple*

```
  tether values = K x y z R0
      K = spring constant (force/distance units)
      x,y,z = point to which spring is tethered
      R0 = equilibrium distance from tether point (distance units)
    couple values = group-ID2 K x y z R0
      group-ID2 = 2nd group to couple to fix group with a spring
      K = spring constant (force/distance units)
      x,y,z = direction of spring
      R0 = equilibrium distance of spring (distance units)
```

Examples:

```
fix pull ligand spring tether 50.0 0.0 0.0 0.0 0.0
fix pull ligand spring tether 50.0 0.0 0.0 0.0 5.0
fix pull ligand spring tether 50.0 NULL NULL 2.0 3.0
fix 5 bilayer1 spring couple bilayer2 100.0 NULL NULL 10.0 0.0
fix longitudinal pore spring couple ion 100.0 NULL NULL -20.0 0.0
fix radial pore spring couple ion 100.0 0.0 0.0 NULL 5.0
```

Description:

Apply a spring force to a group of atoms or between two groups of atoms. This is useful for applying an umbrella force to a small molecule or lightly tethering a large group of atoms (e.g. all the solvent or a large molecule) to the center of the simulation box so that it doesn't wander away over the course of a long simulation. It can also be used to hold the centers of mass of two groups of atoms at a given distance or orientation with respect to each other.

The *tether* style attaches a spring between a fixed point *x,y,z* and the center of mass of the fix group of atoms. The equilibrium position of the spring is R0. At each timestep the distance R from the center of mass of the group of atoms to the tethering point is computed, taking account of wrap-around in a periodic simulation box. A restoring force of magnitude K (R - R0) Mi / M is applied to each atom in the group where *K* is the spring constant, Mi is the mass of the atom, and M is the total mass of all atoms in the group. Note that *K* thus represents the total force on the group of atoms, not a per-atom force.

The *couple* style links two groups of atoms together. The first group is the fix group; the second is specified by group-ID2. The groups are coupled together by a spring that is at equilibrium when the two groups are displaced by a vector *x,y,z* with respect to each other and at a distance R0 from that displacement. Note that *x,y,z* is the equilibrium displacement of group-ID2 relative to the fix group. Thus (1,1,0) is a different spring than (-1,-1,0). When the relative positions and distance between the two groups are not in equilibrium, the same spring force described above is applied to atoms in each of the two groups.

For both the *tether* and *couple* styles, any of the x,y,z values can be specified as NULL which means do not include that dimension in the distance calculation or force application.

The first example above pulls the ligand towards the point (0,0,0). The second example holds the ligand near the surface of a sphere of radius 5 around the point (0,0,0). The third example holds the ligand a distance 3 away from the z=2 plane (on either side).

The fourth example holds 2 bilayers a distance 10 apart in z. For the last two examples, imagine a pore (a slab of atoms with a cylindrical hole cut out) oriented with the pore axis along z, and an ion moving within the pore. The fifth example holds the ion a distance of -20 below the z = 0 center plane of the pore (umbrella sampling). The last example holds the ion a distance 5 away from the pore axis (assuming the center-of-mass of the pore in x,y is the pore axis).

IMPORTANT NOTE: The center of mass of a group of atoms is calculated in "unwrapped" coordinates using atom image flags, which means that the group can straddle a periodic boundary. See the dump doc page for a discussion of unwrapped coordinates. It also means that a spring connecting two groups or a group and the tether point can cross a periodic boundary and its length be calculated correctly. One exception is for rigid bodies, which should not be used with the fix spring command, if the rigid body will cross a periodic boundary. This is because image flags for rigid bodies are used in a different way, as explained on the fix rigid doc page.

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

No information about this fix is written to binary restart files.

The fix_modify *energy* option is supported by this fix to add the energy stored in the spring to the system's potential energy as part of thermodynamic output.

This fix computes a global scalar which can be accessed by various output commands. The scalar is the spring energy = 0.5 * K * r^2.

This fix also computes global 4-vector which can be accessed by various output commands. The first 3 quantities in the vector are xyz components of the total force added to the group of atoms by the spring. In the case of the *couple* style, it is the force on the fix group (group-ID) or the negative of the force on the 2nd group (group-ID2). The 4th quantity in the vector is the magnitude of the force added by the spring, as a positive value if (r-R0) > 0 and a negative value if (r-R0) < 0. This sign convention can be useful when using the spring force to compute a potential of mean force (PMF).

The scalar and vector values calculated by this fix are "extensive".

No parameter of this fix can be used with the *start/stop* keywords of the run command.

The forces due to this fix are imposed during an energy minimization, invoked by the minimize command.

IMPORTANT NOTE: If you want the spring energy to be included in the total potential energy of the system (the quantity being minimized), you MUST enable the fix_modify *energy* option for this fix.

Restrictions: none

Related commands:

fix drag, fix spring/self, fix spring/rg, fix smd

Default: none

fix spring/rg command

Syntax:

```
fix ID group-ID spring/rg K RG0
```

- ID, group-ID are documented in fix command
- spring/rg = style name of this fix command
- K = harmonic force constant (force/distance units)
- RG0 = target radius of gyration to constrain to (distance units)

```
  if RG0 = NULL, use the current RG as the target value
```

Examples:

```
fix 1 protein spring/rg 5.0 10.0
fix 2 micelle spring/rg 5.0 NULL
```

Description:

Apply a harmonic restraining force to atoms in the group to affect their central moment about the center of mass (radius of gyration). This fix is useful to encourage a protein or polymer to fold/unfold and also when sampling along the radius of gyration as a reaction coordinate (i.e. for protein folding).

The radius of gyration is defined as RG in the first formula. The energy of the constraint and associated force on each atom is given by the second and third formulas, when the group is at a different RG than the target value RG0.

$$R_G{}^2 = \frac{1}{M} \sum_i^N m_i \left(x_i - \frac{1}{M} \sum_j^N m_j x_j \right)^2$$

$$E = K \left(R_G - R_{G0} \right)^2$$

$$F_i = 2K \frac{m_i}{M} \left(1 - \frac{R_{G0}}{R_G} \right) \left(x_i - \frac{1}{M} \sum_j^N m_j x_j \right)$$

The (x_i - center-of-mass) term is computed taking into account periodic boundary conditions, m_i is the mass of the atom, and M is the mass of the entire group. Note that K is thus a force constant for the aggregate force on the group of atoms, not a per-atom force.

If RG0 is specified as NULL, then the RG of the group is computed at the time the fix is specified, and that value is used as the target.

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

No information about this fix is written to **binary restart files**. None of the **fix_modify** options are relevant to this fix. No global or per-atom quantities are stored by this fix for access by various **output commands**. No parameter of this fix can be used with the *start/stop* keywords of the **run** command. This fix is not invoked during **energy minimization**.

Restrictions: none

Related commands:

fix spring, fix spring/self fix drag, fix smd

Default: none

fix spring/self command

Syntax:

```
fix ID group-ID spring/self K dir
```

- ID, group-ID are documented in [fix](#) command
- spring/self = style name of this fix command
- K = spring constant (force/distance units)
- dir = xyz, xy, xz, yz, x, y, or z (optional, default: xyz)

Examples:

```
fix tether boundary-atoms spring/self 10.0
fix zrest  move spring/self 10.0 z
```

Description:

Apply a spring force independently to each atom in the group to tether it to its initial position. The initial position for each atom is its location at the time the fix command was issued. At each timestep, the magnitude of the force on each atom is -Kr, where r is the displacement of the atom from its current position to its initial position. The distance r correctly takes into account any crossings of periodic boundary by the atom since it was in its intitial position.

With the (optional) dir flag, one can select in which direction the spring force is applied. By default, the restraint is applied in all directions, but it can be limited to the xy-, xz-, yz-plane and the x-, y-, or z-direction, thus restraining the atoms to a line or a plane, respectively.

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

This fix writes the original coordinates of tethered atoms to [binary restart files](#), so that the spring effect will be the same in a restarted simulation. See the [read_restart](#) command for info on how to re-specify a fix in an input script that reads a restart file, so that the operation of the fix continues in an uninterrupted fashion.

The [fix_modify](#) *energy* option is supported by this fix to add the energy stored in the per-atom springs to the system's potential energy as part of [thermodynamic output](#).

This fix computes a global scalar which can be accessed by various [output commands](#). The scalar is an energy which is the sum of the spring energy for each atom, where the per-atom energy is $0.5 * K * r^2$. The scalar value calculated by this fix is "extensive".

No parameter of this fix can be used with the *start/stop* keywords of the [run](#) command.

The forces due to this fix are imposed during an energy minimization, invoked by the [minimize](#) command.

IMPORTANT NOTE: If you want the per-atom spring energy to be included in the total potential energy of the system (the quantity being minimized), you MUST enable the [fix_modify](#) *energy* option for this fix.

Restrictions: none

Related commands:

fix drag, fix spring, fix smd, fix spring/rg

Default: none

fix store/force command

Syntax:

```
fix ID group-ID store/force
```

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

Examples:

```
fix 1 all store/force
```

Description:

Store the forces on atoms in the group at the point during each timestep when the fix is invoked, as described below. This is useful for storing forces before constraints or other boundary conditions are computed which modify the forces, so that unmodified forces can be [written to a dump file](#) or accessed by other [output commands](#) that use per-atom quantities.

This fix is invoked at the point in the velocity-Verlet timestepping immediately after [pair](#), [bond](#) forces have been calculated. It is the point in the timestep when various fixes that compute constraint forces are calculated and potentially modify the force on each atom. Examples of such fixes are [fix shake](#), [fix wall](#), and [fix indent](#).

IMPORTANT NOTE: The order in which various fixes are applied which operate at the same point during the timestep, is the same as the order they are specified in the input script. Thus normally, if you want to store per-atom forces due to force field interactions, before constraints are applied, you should list this fix first within that set of fixes, i.e. before other fixes that apply constraints. However, if you wish to include certain constraints (e.g. fix shake) in the stored force, then it could be specified after some fixes and before others.

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

No information about this fix is written to [binary restart files](#). None of the [fix_modify](#) options are relevant to this fix.

This fix produces a per-atom array which can be accessed by various [output commands](#). The number of columns for each atom is 3, and the columns store the x,y,z forces on each atom. The per-atom values be accessed on any timestep.

No parameter of this fix can be used with the *start/stop* keywords of the [run](#) command. This fix is not invoked during [energy minimization](#).

Restrictions: none

Related commands:

[fix store_state](#)

Default: none

fix store/state command

Syntax:

```
fix ID group-ID store/state N input1 input2 ... keyword value ...
```

- ID, group-ID are documented in fix command
- store/state = style name of this fix command
- N = store atom attributes every N steps, N = 0 for initial store only
- input = one or more atom attributes

```
possible attributes = id, mol, type, mass,
                      x, y, z, xs, ys, zs, xu, yu, zu, ix, iy, iz,
                      vx, vy, vz, fx, fy, fz,
                      q, mux, muy, muz,
                      radius, omegax, omegay, omegaz,
                      angmomx, angmomy, angmomz, tqx, tqy, tqz
                      c_ID, c_ID[N], f_ID, f_ID[N], v_name

    id = atom ID
    mol = molecule ID
    type = atom type
    mass = atom mass
    x,y,z = unscaled atom coordinates
    xs,ys,zs = scaled atom coordinates
    xu,yu,zu = unwrapped atom coordinates
    ix,iy,iz = box image that the atom is in
    vx,vy,vz = atom velocities
    fx,fy,fz = forces on atoms
    q = atom charge
    mux,muy,muz = orientation of dipolar atom
    radius = radius of spherical particle
    omegax,omegay,omegaz = angular velocity of spherical particle
    angmomx,angmomy,angmomz = angular momentum of aspherical particle
    tqx,tqy,tqz = torque on finite-size particles
    c_ID = per-atom vector calculated by a compute with ID
    c_ID[I] = Ith column of per-atom array calculated by a compute with ID
    f_ID = per-atom vector calculated by a fix with ID
    f_ID[I] = Ith column of per-atom array calculated by a fix with ID
    v_name = per-atom vector calculated by an atom-style variable with name
```

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

```
  com value = yes or no
```

Examples:

```
fix 1 all store/state 0 x y z
fix 1 all store/state 0 xu yu zu com yes
fix 2 all store/state 1000 vx vy vz
```

Description:

Define a fix that stores attributes for each atom in the group at the time the fix is defined. If *N* is 0, then the values are never updated, so this is a way of archiving an atom attribute at a given time for future use in a calculation or output. See the discussion of output commands that take fixes as inputs. And see for example, the compute reduce, fix ave/atom, fix ave/histo, fix ave/spatial, and atom-style variable commands.

If *N* is not zero, then the attributes will be updated every *N* steps.

IMPORTANT NOTE: Actually, only atom attributes specified by keywords like *xu* or *vy* are initially stored immediately at the point in your input script when the fix is defined. Attributes specified by a compute, fix, or variable are not initially stored until the first run following the fix definition begins. This is because calculating those attributes may require quantities that are not defined in between runs.

The list of possible attributes is the same as that used by the dump custom command, which describes their meaning.

If the *com* keyword is set to *yes* then the *xu*, *yu*, and *zu* inputs store the position of each atom relative to the center-of-mass of the group of atoms, instead of storing the absolute position. This option is used by the compute msd command.

The requested values are stored in a per-atom vector or array as discussed below. Zeroes are stored for atoms not in the specified group.

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

This fix writes the per-atom values it stores to binary restart files, so that the values can be restored when a simulation is restarted. See the read_restart command for info on how to re-specify a fix in an input script that reads a restart file, so that the operation of the fix continues in an uninterrupted fashion.

None of the fix_modify options are relevant to this fix.

If a single input is specified, this fix produces a per-atom vector. If multiple inputs are specified, a per-atom array is produced where the number of columns for each atom is the number of inputs. These can be accessed by various output commands. The per-atom values be accessed on any timestep.

No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions: none

Related commands:

dump custom, compute property/atom, variable

Default:

The option default is com = no.

fix viscous command

Syntax:

```
fix ID group-ID viscous gamma keyword values ...
```

- ID, group-ID are documented in fix command
- viscous = style name of this fix command
- gamma = damping coefficient (force/velocity units)
- zero or more keyword/value pairs may be appended

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

Examples:

```
fix 1 flow viscous 0.1
fix 1 damp viscous 0.5 scale 3 2.5
```

Description:

Add a viscous damping force to atoms in the group that is proportional to the velocity of the atom. The added force can be thought of as a frictional interaction with implicit solvent, i.e. the no-slip Stokes drag on a spherical particle. In granular simulations this can be useful for draining the kinetic energy from the system in a controlled fashion. If used without additional thermostatting (to add kinetic energy to the system), it has the effect of slowly (or rapidly) freezing the system; hence it can also be used as a simple energy minimization technique.

The damping force F is given by F = - gamma * velocity. The larger the coefficient, the faster the kinetic energy is reduced. If the optional keyword *scale* is used, gamma can scaled up or down by the specified factor for atoms of that type. It can be used multiple times to adjust gamma for several atom types.

IMPORTANT NOTE: You should specify gamma in force/velocity units. This is not the same as mass/time units, at least for some of the LIGGGHTS(R)-WSU units options like "real" or "metal" that are not self-consistent.

In a Brownian dynamics context, gamma = Kb T / D, where Kb = Boltzmann's constant, T = temperature, and D = particle diffusion coefficient. D can be written as Kb T / (3 pi eta d), where eta = dynamic viscosity of the frictional fluid and d = diameter of particle. This means gamma = 3 pi eta d, and thus is proportional to the viscosity of the fluid and the particle diameter.

In the current implementation, rather than have the user specify a viscosity, gamma is specified directly in force/velocity units. If needed, gamma can be adjusted for atoms of different sizes (i.e. sigma) by using the *scale* keyword.

Note that Brownian dynamics models also typically include a randomized force term to thermostat the system at a chosen temperature. The fix langevin command does this. It has the same viscous damping term as fix viscous and adds a random force to each atom. The random force term is proportional to the sqrt of the chosen thermostatting temperature. Thus if you use fix langevin with a target T = 0, its random force term is zero, and you are essentially performing the same operation as fix viscous. Also note that the gamma of fix viscous is

related to the damping parameter of fix langevin, however the former is specified in units of force/velocity and the latter in units of time, so that it can more easily be used as a thermostat.

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

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix. No global or per-atom quantities are stored by this fix for access by various output commands. No parameter of this fix can be used with the *start/stop* keywords of the run command.

The forces due to this fix are imposed during an energy minimization, invoked by the minimize command. This fix should only be used with damped dynamics minimizers that allow for non-conservative forces. See the min_style command for details.

Restrictions: none

Related commands:

fix langevin

Default: none

fix wall/gran command

Syntax:

```
fix ID group-ID style model_type model_name wallstyle wallstyleargs general_keywords general_valu
```

- ID, group-ID are documented in fix command
- style = *wall/gran*
- zero or more model_type/model_name pairs may be appended. They must be appended in the following order

```
model values = described here
  tangential values = described here
  cohesion values = described here
  rolling_friction values = described here
  surface values = described here
```

- wallstyle = *mesh* or *primitive*
- wallstyle args for wallstyle *mesh* = *n_meshes* and *meshes*

```
n_meshes value = nm
    nm = # of meshes (see fix mesh/surface) to use for the wall (positive integer)
  meshes values = meshlist
    meshlist =  id(s) of the mesh(es) (see fix mesh/surface) to be used. These must be def
```

- wallstyle args for wallstyle *primitive* = *type* or *xplane* or *yplane* or *zplane* or *xcylinder* or *ycylinder* or *zcylinder*

```
type args = tp
    tp = atom_type (material type) of the wall
  xplane or yplane or zplane args = pos
    pos = position plane (distance units)
  xcylinder or ycylinder or zcylinder args = radius c1 c2
    radius = cylinder radius (distance units)
    c1,c2 = coordinates of center axis in other 2 dims (distance units)
```

- zero or more general_keyword/value pairs may be appended
- general_keyword = *shear* or *store_force* or *store_force_contact*

```
shear values = dim vshear
    dim = x or y or z
    vshear = magnitude of shear velocity (velocity units)
  temperature value = T0
    T0 = Temperature of the wall (temperature units)
  contact_area values = 'overlap' or 'constant value' or 'projection'
  store_force value = 'yes' or 'no'
    yes, no = determines if the wall force exerted on the particles is stored in a fix pro
  store_force_contact value = 'yes' or 'no'
    yes, no = determines if the force for each particle-wall contact is stored in a fix pr
```

- following the general_keyword/value pairs, zero or more model_keyword/model_value pairs may be appended in arbitrary order model_keyword/model_value pairs = described for each model separately here

Examples:

```
fix zwalls all wall/gran model hertz tangential history primitive type 1 zplane 0.15
fix zwalls all wall/gran model hertz tangential history primitive type 1 zplane 0.15 surface supe
fix meshwalls all wall/gran model hertz tangential history mesh n_meshes 2 meshes cad1 cad2
```

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 equation for the force between the wall and particles touching it is the same as the corresponding equation on the pair_style granular doc page, in the limit of one of the two particles going to infinite radius and mass (flat wall).

You must choose the models matching the pair style used, otherwise an error is created. As with pair_style granular, you have to define the mechanical properties for each material you are used in the simulation with fix property commands. See pair_style gran for more details and the model doc page for details.

For wallstyle *mesh*, fix_id1, fix_id2 etc. must be IDs of valid fixes of type fix mesh/surface, defining the granular mesh to be used for the wall. Triangle-particle neighbor lists are built to efficiently track particle-triangle contacts. Particle-tri neighbor list build is triggered if any particle moves more than half the skin distance or (in case of moving mesh) if the mesh itself moves more than half the skin distance since the last build. A warning is generated if a dangerous particle-tri neigh list build is detected (e.g. if particles are inserted too close to a wall, see section 'Restrictions'). For style *mesh*, the atom_type (material type) is inherited from the atom style provided in the fix mesh/surface command.

For *wallstyle primitive*, the atom_type (material type) has to be provided via keyword *type*. Primitve walls can be *xplane* or *yplane* or *zplane* or *cylindrical*. The 3 planar options specify a single wall in a dimension. Wall positions are given by values for lo and hi. For an *xcylinder*, *ycylinder* or *zcylinder*, the radius and the cylinder axis in the other two dims is specified.

Optionally, primitive walls can be moving, if the shear keyword is appended.

For the *shear* keyword, the wall moves continuously in the specified dimension with velocity vshear. The dimension must be tangential to walls with a planar wallstyle, e.g. in the y or z directions for an *xplane* wall. For *zcylinder* walls, a dimension of z means the cylinder is moving in the z-direction along it's axis. A dimension of x or y means the cylinder is spinning around the z-axis, either in the clockwise direction for vshear > 0 or counter-clockwise for vshear < 0. In this case, vshear is the tangential velocity of the wall at whatever radius has been defined. The same applies to *xcylinder* and *ycylinder* accordingly.

NOTE: The keywords wiggle or shear can NOT be used for *wallstyle mesh*. For a moving a granular wall with *wallstyle mesh*, use the more flexible command "fix move/mesh"fix_move_mesh.html, or use the keywords *velocity* or *angular_velocity* in fix mesh/surface.

The keyword *temperature* is used to assign a constant temperature to the wall. This setting gets effective in conjunction with heat conduction via fix heat/gran/conduction. For *wallstyle mesh*, the value for the temperature given in this command is ignored and the temperature value is specified per mesh via fix mesh/surface. Contact area calculation can be controlled by keyword *contact_area* in the same manner as for fix heat/gran/conduction.

By specifying *store_force* = 'yes', you can instruct the command to store the wall force exerted on the particles in a fix property/atom with id force_(ID), where (ID) is the id of the fix wall/gran command.

store_force_contact does the same for contact forces between particle and wall.

The effect of keyword *rolling_friction*, *cohesion*, *tangential_damping*, *viscous* and *absolute_damping* is explanted in pair gran

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

If applicable, contact history is written to binary restart files so simulations can continue properly. None of the fix_modify options are relevant to this fix. No global scalar or vector or are stored by this fix. If *store_force* = 'yes' is specified, the per-atom wall force can be accessed by the various output commands via f_force(ID)**1**,

f_force(ID)**2**, f_force(ID)**3**. (ID) is the id of the fix wall/gran command. No parameter of this fix can be used with the start/stop keywords of the [run](#) command. This fix is not invoked during [energy minimization](#) .

Restrictions:

There can be only one fix wall/gran command with style *mesh*. Note that this is not really a restriction because you can include multiple fixes of type [fix mesh/surface](#) in the fix wall/gran command.

store_force_contact is available in selected versions only; it is not available in the PUBLIC version.

When using style style *mesh*, you have to use the style *bin* for the [neighbor command](#).

Style *mesh* can not be used in conjunction with triclinic simulation boxes.

When using style *mesh*' in combination with a particle insertion command, you always have to keep a minimum distance between the wall and the insertion region that is equal to maximum particle radius + half the skin defined in the [neighbor command](#). Otherwise, newly inserted particles interpenetrate the walls before a triangle neighbor list is built the first time.

The keyword shear can NOT be used for style *mesh*. For moving granular wall with style mesh, use "fix move/mesh"fix_move_mesh.html.

Any dimension (xyz) that has a planar granular wall must be non-periodic.

Related commands:

[fix mesh/surface](#), [fix_move_mesh](#), [pair_style granular](#) Models for use with this command are described [here](#)

Default:

model = 'hertz' *tangential* = 'history' *rolling_friction* = 'off' *cohesion* = 'off' *surface* = 'default'

fix wall/reflect command

Syntax:

```
fix ID group-ID wall/reflect face arg ... keyword value ...
```

- ID, group-ID are documented in [fix](#) command
- wall/reflect = style name of this fix command
- one or more face/arg pairs may be appended
- face = *xlo* or *xhi* or *ylo* or *yhi* or *zlo* or *zhi*

```
xlo,ylo,zlo arg = EDGE or constant or variable
    EDGE = current lo edge of simulation box
    constant = number like 0.0 or -30.0 (distance units)
    variable = equal-style variable like v_x or v_wiggle
  xhi,yhi,zhi arg = EDGE or constant or variable
    EDGE = current hi edge of simulation box
    constant = number like 50.0 or 100.3 (distance units)
    variable = equal-style variable like v_x or v_wiggle
```

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

```
units value = lattice or box
    lattice = the wall position is defined in lattice units
    box = the wall position is defined in simulation box units
```

Examples:

```
fix xwalls all wall/reflect xlo EDGE xhi EDGE
fix walls all wall/reflect xlo 0.0 ylo 10.0 units box
fix top all wall/reflect zhi v_pressdown
```

Description:

Bound the simulation with one or more walls which reflect particles in the specified group when they attempt to move thru them.

Reflection means that if an atom moves outside the wall on a timestep by a distance delta (e.g. due to [fix nve](#)), then it is put back inside the face by the same delta, and the sign of the corresponding component of its velocity is flipped.

When used in conjunction with [fix nve](#) and [run_style verlet](#), the resultant time-integration algorithm is equivalent to the primitive splitting algorithm (PSA) described by [Bond](#). Because each reflection event divides the corresponding timestep asymmetrically, energy conservation is only satisfied to O(dt), rather than to O(dt^2) as it would be for velocity-Verlet integration without reflective walls.

Up to 6 walls or faces can be specified in a single command: *xlo*, *xhi*, *ylo*, *yhi*, *zlo*, *zhi*. A *lo* face reflects particles that move to a coordinate less than the wall position, back in the *hi* direction. A *hi* face reflects particles that move to a coordinate higher than the wall position, back in the *lo* direction.

The position of each wall can be specified in one of 3 ways: as the EDGE of the simulation box, as a constant value, or as a variable. If EDGE is used, then the corresponding boundary of the current simulation box is used. If a numeric constant is specified then the wall is placed at that position in the appropriate dimension (x, y, or z). In both the EDGE and constant cases, the wall will never move. If the wall position is a variable, it

should be specified as v_name, where name is an [equal-style variable](#) name. In this case the variable is evaluated each timestep and the result becomes the current position of the reflecting wall. Equal-style variables can specify formulas with various mathematical functions, and include [thermo_style](#) command keywords for the simulation box parameters and timestep and elapsed time. Thus it is easy to specify a time-dependent wall position.

The *units* keyword determines the meaning of the distance units used to define a wall position, but only when a numeric constant or variable is used. It is not relevant when EDGE is used to specify a face position. In the variable case, the variable is assumed to produce a value compatible with the *units* setting you specify.

A *box* value selects standard distance units as defined by the [units](#) command, e.g. Angstroms for units = real or metal. A *lattice* value means the distance units are in lattice spacings. The [lattice](#) command must have been previously used to define the lattice spacings.

Here are examples of variable definitions that move the wall position in a time-dependent fashion using equal-style [variables](#).

```
variable ramp equal ramp(0,10)
fix 1 all wall/reflect xlo v_ramp

variable linear equal vdisplace(0,20)
fix 1 all wall/reflect xlo v_linear

variable wiggle equal swiggle(0.0,5.0,3.0)
fix 1 all wall/reflect xlo v_wiggle

variable wiggle equal cwiggle(0.0,5.0,3.0)
fix 1 all wall/reflect xlo v_wiggle
```

The ramp(lo,hi) function adjusts the wall position linearly from lo to hi over the course of a run. The vdisplace(c0,velocity) function does something similar using the equation position = c0 + velocity*delta, where delta is the elapsed time.

The swiggle(c0,A,period) function causes the wall position to oscillate sinusoidally according to this equation, where omega = 2 PI / period:

```
position = c0 + A sin(omega*delta)
```

The cwiggle(c0,A,period) function causes the wall position to oscillate sinusoidally according to this equation, which will have an initial wall velocity of 0.0, and thus may impose a gentler perturbation on the particles:

```
position = c0 + A (1 - cos(omega*delta))
```

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

No information about this fix is written to [binary restart files](#). None of the [fix_modify](#) options are relevant to this fix. No global or per-atom quantities are stored by this fix for access by various [output commands](#). No parameter of this fix can be used with the *start/stop* keywords of the [run](#) command. This fix is not invoked during [energy minimization](#).

Restrictions:

Any dimension (xyz) that has a reflecting wall must be non-periodic.

fix wall/reflect command

A reflecting wall should not be used with rigid bodies such as those defined by a "fix rigid" command. This is because the wall/reflect displaces atoms directly rather than exerts a force on them. For rigid bodies, use a soft wall instead, such as [fix wall/lj93](). LIGGGHTS(R)-WSU will flag the use of a rigid fix with fix wall/reflect with a warning, but will not generate an error.

Related commands:

[fix wall/lj93]() command

Default:

units = box

(Bond) Bond and Leimkuhler, SIAM J Sci Comput, 30, p 134 (2007).

fix wall/region command

Syntax:

```
fix ID group-ID wall/region region-ID style epsilon sigma cutoff
```

- ID, group-ID are documented in [fix](#) command
- wall/region = style name of this fix command
- region-ID = region whose boundary will act as wall
- style = *lj93* or *lj126* or *colloid* or *harmonic*
- epsilon = strength factor for wall-particle interaction (energy or energy/distance^2 units)
- sigma = size factor for wall-particle interaction (distance units)
- cutoff = distance from wall at which wall-particle interaction is cut off (distance units)

Examples:

```
fix wall all wall/region mySphere lj93 1.0 1.0 2.5
```

Description:

Treat the surface of the geometric region defined by the *region-ID* as a bounding wall which interacts with nearby particles according to the specified style. The distance between a particle and the surface is the distance to the nearest point on the surface and the force the wall exerts on the particle is along the direction between that point and the particle, which is the direction normal to the surface at that point.

Regions are defined using the [region](#) command. Note that the region volume can be interior or exterior to the bounding surface, which will determine in which direction the surface interacts with particles, i.e. the direction of the surface normal. Regions can either be primitive shapes (block, sphere, cylinder, etc) or combinations of primitive shapes specified via the *union* or *intersect* region styles. These latter styles can be used to construct particle containers with complex shapes. Regions can also change over time via the [region](#) command keywords (move) and *rotate*. If such a region is used with this fix, then the of region surface will move over time in the corresponding manner.

IMPORTANT NOTE: As discussed on the [region](#) command doc page, regions in LIGGGHTS(R)-WSU do not get wrapped across periodic boundaries. It is up to you to insure that periodic or non-periodic boundaries are specified appropriately via the [boundary](#) command when using a region as a wall that bounds particle motion. This also means that if you embed a region in your simulation box and want it to repulse particles from its surface (using the "side out" option in the [region](#) command), that its repulsive force will not be felt across a periodic boundary.

IMPORTANT NOTE: For primitive regions with sharp corners and/or edges (e.g. a block or cylinder), wall/particle forces are computed accurately for both interior and exterior regions. For *union* and *intersect* regions, additional sharp corners and edges may be present due to the intersection of the surfaces of 2 or more primitive volumes. These corners and edges can be of two types: concave or convex. Concave points/edges are like the corners of a cube as seen by particles in the interior of a cube. Wall/particle forces around these features are computed correctly. Convex points/edges are like the corners of a cube as seen by particles exterior to the cube, i.e. the points jut into the volume where particles are present. LIGGGHTS(R)-WSU does NOT compute the location of these convex points directly, and hence wall/particle forces in the cutoff volume around these points suffer from inaccuracies. The basic problem is that the outward normal of the surface is not continuous at these points. This can cause particles to feel no force (they don't "see" the wall) when in one location, then move a distance epsilon, and suddenly feel a large force because they now "see" the wall. In the

worst-case scenario, this can blow particles out of the simulation box. Thus, as a general rule you should not use the fix wall/region command with *union* or *interesect* regions that have convex points or edges.

The energy of wall-particle interactions depends on the specified style.

For style *lj93*, the energy E is given by the 9/3 potential:

$$E = \epsilon \left[\frac{2}{15} \left(\frac{\sigma}{r} \right)^9 - \left(\frac{\sigma}{r} \right)^3 \right] \qquad r < r_c$$

For style *lj126*, the energy E is given by the 12/6 potential:

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

For style *colloid*, the energy E is given by an integrated form of the pair_style colloid potential:

$$E = \epsilon \left[\frac{\sigma^6}{7560} \left(\frac{6R - D}{D^7} + \frac{D + 8R}{(D + 2R)^7} \right) - \frac{1}{6} \left(\frac{2R(D + R) + D(D + 2R) \left[\ln D - \ln(D + 2R) \right]}{D(D + 2R)} \right) \right] \qquad r < r$$

For style *wall/harmonic*, the energy E is given by a harmonic spring potential:

$$E = \epsilon \left(r - r_c \right)^2 \qquad r < r_c$$

In all cases, *r* is the distance from the particle to the region surface, and Rc is the *cutoff* distance at which the particle and surface no longer interact. The energy of the wall potential is shifted so that the wall-particle interaction energy is 0.0 at the cutoff distance.

For the *lj93* and *lj126* styles, *epsilon* and *sigma* are the usual Lennard-Jones parameters, which determine the strength and size of the particle as it interacts with the wall. Epsilon has energy units. Note that this *epsilon* and *sigma* may be different than any *epsilon* or *sigma* values defined for a pair style that computes particle-particle interactions.

The *lj93* interaction is derived by integrating over a 3d half-lattice of Lennard-Jones 12/6 particles. The *lj126* interaction is effectively a harder, more repulsive wall interaction.

For the *colloid* style, *epsilon* is effectively a Hamaker constant with energy units for the colloid-wall interaction, *R* is the radius of the colloid particle, *D* is the distance from the surface of the colloid particle to the wall (r-R), and *sigma* is the size of a constituent LJ particle inside the colloid particle. Note that the cutoff distance Rc in this case is the distance from the colloid particle center to the wall.

The *colloid* interaction is derived by integrating over constituent LJ particles of size *sigma* within the colloid particle and a 3d half-lattice of Lennard-Jones 12/6 particles of size *sigma* in the wall.

For the *wall/harmonic* style, *epsilon* is effectively the spring constant K, and has units (energy/distance^2). The input parameter *sigma* is ignored. The minimum energy position of the harmonic spring is at the *cutoff*. This is a repulsive-only spring since the interaction is truncated at the *cutoff*

IMPORTANT NOTE: For all of the styles, you must insure that r is always > 0 for all particles in the group, or LIGGGHTS(R)-WSU will generate an error. This means you cannot start your simulation with particles on the region surface (r = 0) or with particles on the wrong side of the region surface (r < 0). For the *wall/lj93* and *wall/lj126* styles, the energy of the wall/particle interaction (and hence the force on the particle) blows up as r -> 0. The *wall/colloid* style is even more restrictive, since the energy blows up as D = r-R -> 0. This means the finite-size particles of radius R must be a distance larger than R from the region surface. The *harmonic* style is a softer potential and does not blow up as r -> 0, but you must use a large enough *epsilon* that particles always reamin on the correct side of the region surface (r > 0).

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

No information about this fix is written to binary restart files.

The fix_modify *energy* option is supported by this fix to add the energy of interaction between atoms and the wall to the system's potential energy as part of thermodynamic output.

This fix computes a global scalar energy and a global 3-length vector of forces, which can be accessed by various output commands. The scalar energy is the sum of energy interactions for all particles interacting with the wall represented by the region surface. The 3 vector quantities are the x,y,z components of the total force acting on the wall due to the particles. The scalar and vector values calculated by this fix are "extensive".

No parameter of this fix can be used with the *start/stop* keywords of the run command.

The forces due to this fix are imposed during an energy minimization, invoked by the minimize command.

IMPORTANT NOTE: If you want the atom/wall interaction energy to be included in the total potential energy of the system (the quantity being minimized), you MUST enable the fix_modify *energy* option for this fix.

Restrictions: none

Related commands:

fix wall/lj93, fix wall/lj126, fix wall/colloid, fix wall/gran

Default: none

githubAccess_non-public

Description:

This routine describes how to setup a github account and pull repositories of the CFDEMproject.

Procedure:

Basically the following steps have to be performed:

- get access to non-public repositories
- create an account at http://github.com
- create your RSA key
- add your RSA key to your github account
- *git clone* the desired repository
- update your repositories by *git pull*

Get acces to non-public repositories:

If you have a support contract / non-public repository access by DCS Computing GmbH, please follow the steps below to set up your user and RSA key. After that please send your username and company affiliation to DCS Computing GmbH to get your account activated. Afterwards you can clone also the non-public repositories.

Create an account:

Please create a free account at https://github.com.

```
example:
user (username)
user@mail.com
pwd (pwd)
```

Please use your own username and mail adress here and for the following steps!

Create your RSA key:

Please find the complete setup description here, or use the short description below.

Open a terminal and execute:

```
cd ~/.ssh
ssh-keygen -t rsa -C "user@mail.com"
gedit id_rsa.pub&
```

Add your RSA key to your github account:

Login at https://github.com with you user, then

- click *Account Settings*
- click *SSH Keys*
- click *Add SSH key*
- paste your key into the *Key* field

• Hit *Add Key.*

To check your settings, open a terminal and execute:

```
ssh -T git@github.com
```

git clone the desired repository:

To clone the non-public LIGGGHTS(R)-WSU repository (in this example LIGGGHTS-COMPANY, where COMPANY is the name of your company), open a terminal and execute:

```
git clone git@github.com:CFDEMproject/LIGGGHTS-COMPANY.git
```

Note: the git protocol will not work if your computer is behind a firewall which blocks the relevant TCP port, you can use alternatively this command (you need to enter your password):

```
git clone https://user@github.com/CFDEMproject/LIGGGHTS-COMPANY.git
```

Update your repositories by *git pull*:

To get the latest version, open a terminal, go to the location of your local installation and type:

```
git pull
```

githubAccess_public

Description:

This routine describes how to setup a github account and pull repositories of the CFDEMproject.

Procedure:

- Basically the following steps have to be performed: *git clone* the desired repository
- update your repositories by *git pull*

git clone **the desired repository:**

To clone the public LIGGGHTS(R)-WSU repository, open a terminal and execute:

```
git clone git@github.com:CFDEMproject/LIGGGHTS-PUBLIC.git
```

Note: the git protocol will not work if your computer is behind a firewall which blocks the relevant TCP port, you can use alternatively:

```
git clone https://github.com/CFDEMproject/LIGGGHTS-PUBLIC.git
```

To clone the public CFDEMcoupling repository, open a terminal and execute:

```
git clone git@github.com:CFDEMproject/CFDEMcoupling-PUBLIC.git
```

Note: the git protocol will not work if your computer is behind a firewall which blocks the relevant TCP port, you can use alternatively:

```
git clone https://github.com/CFDEMproject/CFDEMcoupling-PUBLIC.git
```

Update your repositories by *git pull*:

To get the latest version, open a terminal, go to the location of your local installation and type:

```
git pull
```

gran cohesion easo/capillary/viscous

Syntax:

```
cohesion easo/capillary/viscous
```

Description:

This model can be used as part of pair gran It adds a liquid bridge force, caused by a surface liquid film on the particles, to a pair of particles, which consists of a capilary and a viscous part. Furthermore, it solves for the transfer of surface liquid from one particle to the other as the bridge breaks up. The model uses a parameter, *maxSeparationDistanceRatio*, to apply a cut-off to the liquid bridge force, i.e. *radius*maxSeparationDistanceRatio* is the effective contact radius of the particles. The model follows a composition of models suggested by (Easo)

Bridge formation and break-up, surface liquid transfer:

Vbond, the volume of surface liquid involved in the bridge, is given by

```
Vbond = 0.05*(surfaceLiquidVolI+surfaceLiquidVolJ)
```

where *surfaceLiquidVolI/J* is the surface liquid volume attached to particle i/j. This model assumes that bridge formation occurs upon contact and the rupture distance is given as follows by (Lian)

```
dist0 = (1+contactAngleEff/2)*Vbond^(1/3)
```

When the bridge breaks, it is assumed that the surface liquid volume distributes evenly to the two particles.

Capilary force:

The capilary force is given by (Soulie) as:

```
Fcapilary = - pi*surfaceTension*sqrt(radi*radj)*(exp(A*dist/R2+B)+C)
```

where

```
volBondScaled = volBond/R2^3
A = -1.1*volBondScaled^(-0.53)
B = (-0.148*log(volBondScaled)-0.96)*contactAngleEff*contactAngleEff - 0.0082*log(volBondScaled)
C = 0.0018*log(volBondScaled)+0.078;
contactAngleEff = 0.5*(contactAngleI+contactAngleJ)
```

dist is the distance between the particles' surfaces, *surfaceTension* is the surface tension of the fluid, *contactAngleI/J* are the contact angles for particle i/j and the fluid. *R2* is the radius of the larger of the two particles in contact.

Viscous force:

the normal and tangential parts of the viscous force are calculated as given by (Nase)

```
rEff = radi*radj/(radi+radj)
stokesPreFactor = 6*pi*fluidViscosity*rEff
FviscN = stokesPreFactor*vn*rEff/dist
```

```
FviscT = stokesPreFactor*vt*(8/15*log(rEff/dist)+0.9588)
```

where *vn* and *vt* are the normal and tangential relative velocities of the particles at the contact point, *fluidViscosity* is the viscosity of the fluid and *ri* and *rj* are the particle radii. An additional parameter, *minSeparationDistanceRatio*, is used to prevent the value of the viscous force from becoming too large, i.e. *radius*minSeparationDistanceRatio* is assumed to be the minimum separation distance.

Computation of liquid transport and effect of liquid content on other particle properties:

Per default, this model automatically instatiates a scalar transport equation that solved for the surface liquid content of each particles, expressed in volume % of solid volume (4/3 pi * radius ^3). The surface liquid volume is assumed to be small, i.e. it is assumed to have no effect on the particle mass, diameter and density.

The user can override the default behavior by explicity specifing a fix that solves for the surface liquid transport between particles. Such fixes are fix liquidtransport/porous or fix liquidtransport/sponge

---

Intialization:

```
If you are using the this model model, you must define the following properties:
fix id all property/global minSeparationDistanceRatio scalar value
    (value=value for the minimum separation distance, recommended as 1.01)
fix id all property/global maxSeparationDistanceRatio scalar value
    (value=value for the maximum separation distance, recommended as 1.1)
fix id all property/global surfaceLiquidContentInitial scalar value
    (value=value for the initial surface liquid volume in % of the solid volume)
fix id all property/global surfaceTension scalar value
    (value=value for the surface tension of liquid (SI units N/m))
fix id all property/global fluidViscosity scalar value
    (value=value for the fluidViscosity (SI units Pas))
fix id all property/global contactAngle peratomtype value_1 value_2 ...
    (value_i=value for contact angle of atom type i and fluid)
```

Restrictions:

This model can ONLY be used as part of pair gran, not as part of a fix wall/gran.

Output:

This gran model stores a couple of per-particle properties, for access by various output commands.

You can access the property surfaceLiquidContent by f_surfaceLiquidContent[0] (units % of solid particle volume), liquidFlux (units % of solid particle volume/time) by accessing f_liquidFlux[0] and liquidSource (units % of solid particle volume/time) by accessing f_liquidSource[0]. The latter can be used to manually set a surface liquid source via the set command.

Currently, there is a restriction that these properties can only be accessed after a run 0 command.

Restrictions:

The cohesion model is not available for atom_style superquadric

References:

(Easo) Easo, Wassgreen, Comparison of Liquid Bridge Volume Models in DEM Simulations, AIChE Annual Conference (2013).

gran cohesion easo/capillary/viscous

(Lian) Lian, Thornton, Adams, Journal of Colloid and Interface Science, p134, 161 (1993).

(Nase) S. T. Nase, W. L. Vargas, A. A. Abatan, and J. J. Mc-carthy, Powder Technol 116, 214 (2001).

(Shi) Shi, McCarthy, PowderTechnology, p64, 184 (2008)

(Soulie) Soulie, Cherblanc, Youssoufi, Saix, Intl. J Numerical and Analytical Methods in Geomechanics, p213, 30 (2006)

gran cohesion sjkr2

Syntax:

```
cohesion sjkr2
```

Description:

This model can be used as part of pair gran and fix wall/gran

The modified simplified JKR - Johnson-Kendall-Roberts (SJKR2) model adds an additional normal force contribution. If two particle are in contact, it adds an additional normal force tending to maintain the contact, which writes

F = k A,

where A is the particle contact area and k is the cohesion energy density in J/m3. For *sjkr2*, the sphere-sphere contact area is calculated as

A = 2*Pi * delta_n * (2R*)

If you are using the SJKR2 model, you must also define the cohesion energy density:

```
fix id all property/global cohesionEnergyDensity peratomtypepair n_atomtypes value_11 value_12 ..
    (value_ij=value for the cohesion energy density (in Energy/Length3 units) between atom type i
```

IMPORTANT NOTE: The cohesion model has been derived for the Hertzian Style, it may note be appropriate for the Hookean styles.

IMPORTANT NOTE: You have to use atom styles beginning from 1, e.g. 1,2,3,...

Restrictions:

The cohesion model has been derived for the Hertzian Style, it may note be appropriate for the Hookean styles.

It is not available for atom_style superquadric

gran cohesion sjkr

Syntax:

```
cohesion sjkr
```

Description:

This model can be used as part of pair gran and fix wall/gran

The simplified JKR - Johnson-Kendall-Roberts (SJKR) model adds an additional normal force contribution. If two particle are in contact, it adds an additional normal force tending to maintain the contact, which writes

F = k A,

where A is the particle contact area and k is the cohesion energy density in J/m3. For *sjkr*, the sphere-sphere contact area is calculated as (http://mathworld.wolfram.com/Sphere-SphereIntersection.html)

A = Pi/4 * ((dist-Ri-Rj)*(dist+Ri-Rj)*(dist-Ri+Rj)*(dist+Ri+Rj))/(dist*dist)

where dist is the distance between the particle centers.

If you are using the SJKR model, you must also define the cohesion energy density:

```
fix id all property/global cohesionEnergyDensity peratomtypepair n_atomtypes value_11 value_12 ..
    (value_ij=value for the cohesion energy density (in Energy/Length3 units) between atom type i
```

IMPORTANT NOTE: You have to use atom styles beginning from 1, e.g. 1,2,3,...

Restrictions:

The cohesion model has been derived for the Hertzian Style, it may note be appropriate for the Hookean styles.

It is not available for atom_style superquadric

gran cohesion washino/capillary/viscous

Syntax:

```
cohesion washino/capillary/viscous
```

Description:

This model can be used as part of pair gran It adds a liquid bridge force, caused by a surface liquid film on the particles, to a pair of particles, which consists of a capilary and a viscous part. Furthermore, it solves for the transfer of surface liquid from one particle to the other as the bridge breaks up. The model uses a parameter, *maxSeparationDistanceRatio*, to apply a cut-off to the liquid bridge force, i.e. *radius*maxSeparationDistanceRatio* is the effective contact radius of the particles.

The model parameter *maxLiquidContent* allows to limit the maximum per particle liquid content.

Bridge formation and break-up, surface liquid transfer:

Vbond, the volume of surface liquid involved in the bridge, is given by

```
Vbond = 0.05*(surfaceLiquidVolI+surfaceLiquidVolJ)
```

where *surfaceLiquidVolI/J* is the surface liquid volume attached to particle i/j. This model assumes that both formation distance and rupture distance are given as follows by (Lian)

```
dist0 = (1+contactAngleEff/2)*Vbond^(1/3)
```

When the bridge breaks, it is assumed that the surface liquid volume distributes evenly to the two particles.

Capilary force:

The capilary force is given by (Rabinovitch) as

```
Fcapilary = 2*pi*rEff*surfaceTension* (cos(contactAngleEff) / (1. + dist/(2.*dSpSp)) + sin(alpha)
```

where

```
prefactor = -1+sqrt(1+2*volBond/(pi*rEff*dist^2))
dSpSp = 0.5*dist*prefactor
alpha = sqrt(dist/rEff*prefactor)
contactAngleEff = 0.5*(contactAngleI+contactAngleJ)
```

dist is the distance between the particles' surfaces, *surfaceTension* is the surface tension of the fluid, *contactAngleI/J* are the contact angles for particle i/j and the fluid.

Viscous force:

the normal and tangential parts of the viscous force are calculated as given by (Nase)

```
rEff = radi*radj/(radi+radj)
stokesPreFactor = 6*pi*fluidViscosity*rEff
FviscN = stokesPreFactor*vn*rEff/dist
FviscT = stokesPreFactor*vt*(8/15*log(rEff/dist)+0.9588)
```

where *vn* and *vt* are the normal and tangential relative velocities of the particles at the contact point, *fluidViscosity* is the viscosity of the fluid and *ri* and *rj* are the particle radii. An additional parameter, *minSeparationDistanceRatio*, is used to prevent the value of the viscous force from becoming too large, i.e. *radius*minSeparationDistanceRatio* is assumed to be the minimum separation distance.

Computation of liquid transport and effect of liquid content on other particle properties:

Per default, this model automatically instatiates a scalar transport equation that solved for the surface liquid content of each particles, expressed in volume % of solid volume (4/3 pi * radius ^3). The surface liquid volume is assumed to be small, i.e. it is assumed to have no effect on the particle mass, diameter and density.

The user can override the default behavior by explicity specifing a fix that solves for the surface liquid transport between particles. Such fixes are fix liquidtransport/porous or fix liquidtransport/sponge

---

Intialization:

```
If you are using the this model model, you must define the following properties:
fix id all property/global minSeparationDistanceRatio scalar value
    (value=value for the minimum separation distance, recommended as 1.01)
fix id all property/global maxSeparationDistanceRatio scalar value
    (value=value for the maximum separation distance, recommended as 1.1)
fix id all property/global surfaceLiquidContentInitial scalar value
    (value=value for the initial surface liquid volume in % of the solid volume)
fix id all property/global surfaceTension scalar value
    (value=value for the surface tension of liquid (SI units N/m))
fix id all property/global fluidViscosity scalar value
    (value=value for the fluidViscosity (SI units Pas))
fix id all property/global contactAngle peratomtype value_1 value_2 ...
    (value_i=value for contact angle of atom type i and fluid)
fix id all property/global maxLiquidContent peratomtype value_1 value_2 ...
    (value_i=value for contact angle of atom type i and fluid)
```

Output:

This gran model stores a couple of per-particle properties, for access by various output commands.

You can access the property surfaceLiquidContent by f_surfaceLiquidContent[0] (units % of solid particle volume), liquidFlux (units % of solid particle volume/time) by accessing f_liquidFlux[0] and liquidSource (units % of solid particle volume/time) by accessing f_liquidSource[0]. The latter can be used to manually set a surface liquid source via the set command.

Currently, there is a restriction that these properties can only be accessed after a run 0 command.

Restrictions:

This model can ONLY be used as part of pair gran, not as part of a fix wall/gran.

The cohesion model is not available for atom_style superquadric

References:

(Lian) Lian, Thornton, Adams, Journal of Colloid and Interface Science, p134, 161 (1993).

(Nase) S. T. Nase, W. L. Vargas, A. A. Abatan, and J. J. Mc-carthy, Powder Technol 116, 214 (2001).

gran cohesion washino/capillary/viscous

(Rabinovitch) Rabinovitch, Esayanur, Moudil, Langmuir, p10992, 21 (2005)

---

gran model hertz

Syntax:

```
model hertz [other model_type/model_name pairs as described here ] keyword values
```

• zero or more keyword/value pairs may be appended to the end (after all models are specified)

```
  limitForce values = 'on' or 'off'
     on = ensures that the normal force is never attractive (an artefact that can occur at the e
     off = standard implementation that might lead to attractive forces.
   tangential_damping values = 'on' or 'off'
     on = activates tangential damping
     off = no tangential damping
  heating_normal_hertz values = 'on' or 'off'
     on = model contributes to surface heating in the frame of surface_sphere/heatable
     off = model does not contributes to surface heating
```

Description:

This granular model uses the following formula for the frictional force between two granular particles, when the distance r between two particles of radii Ri and Rj is less than their contact distance d = Ri + Rj. There is no force between the particles when r > d:

$$F = (\underbrace{k_n\ \underbrace{\delta n_{ij}}_{\substack{normal\\overlap}} - \gamma_n\ \underbrace{vn_{ij}}_{\substack{normal\\relative\ vel.}}}_{normal\ force}) + (\underbrace{k_t\ \underbrace{\delta t_{ij}}_{\substack{tangential\\overlap}} - \gamma_t\ \underbrace{vt_{ij}}_{\substack{tangential\\relative\ v}}}_{tangential\ force}$$

The tangential overlap is truncated to fulfil $F_t \le X_\mu F_n$

In the first term is the normal force between the two particles and the second term is the tangential force. The normal force has 2 terms, a spring force and a damping force. The tangential force also has 2 terms: a shear force and a damping force. The shear force is a "history" effect that accounts for the tangential displacement ("tangential overlap") between the particles for the duration of the time they are in contact. This term is controlled by the tangential model in action Keyword *tangential_damping* can be used to eliminate the second part of the force in tangential direction. The way how the Coulomb friction limit acts is also controlled by the tangential model chosen by the user.

The quantities in the equations are as follows:

- delta_n = d - r = overlap distance of 2 particles
- k_n = elastic constant for normal contact

- k_t = elastic constant for tangential contact
- gamma_n = viscoelastic damping constant for normal contact
- gamma_t = viscoelastic damping constant for tangential contact
- delta_t = tangential displacement vector between 2 spherical particles which is truncated to satisfy a frictional yield criterion
- rmu = coefficient of rolling friction
- contactradius = contact radius, equal to particle radius - 0.5 * delta_n
- v_n = normal component of the relative velocity of the 2 particles
- v_t = tangential component of the relative velocity of the 2 particles
- w_r = relative rotational velocity of the 2 particles

The Kn, Kt, gamma_n, and gamma_t coefficients are calculated as follows from the material properties:

$$k_n = \frac{4}{3} Y^* \sqrt{R^* \delta_n} \, ,$$

$$\gamma_n = -2 \sqrt{\frac{5}{6}} \, \beta \sqrt{S_n m^*} \geq 0,$$

$$k_t = 8 G^* \sqrt{R^* \delta_n} \, ,$$

$$\gamma_t = -2 \sqrt{\frac{5}{6}} \, \beta \sqrt{S_t m^*} \geq 0.$$

$$S_n = 2 Y^* \sqrt{R^* \delta_n} \, , \qquad S_t = 8 G^* \sqrt{R^* \delta_n}$$

$$\beta = \frac{ln(e)}{\sqrt{ln^2(e) + \pi^2}} \, ,$$

$$\frac{1}{Y^*} = \frac{\left(1 - v_1^2\right)}{Y_1} + \frac{\left(1 - v_2^2\right)}{Y_2} \, ,$$

$$\frac{1}{G^*} = \frac{2\left(2 - v_1\right)\left(1 + v_1\right)}{Y_1} + \frac{2\left(2 - v_2\right)\left(1 + v_2\right)}{Y_2}$$

$$\frac{1}{R^*} = \frac{1}{R_1} + \frac{1}{R_2} \, , \quad \frac{1}{m^*} = \frac{1}{m_1} + \frac{1}{m_2}$$

Y…Young's modulus   G…Shear modulus

v…Poisson ratio   e…coeff. of restitution

To define those material properties, it is mandatory to use multiple fix property/global commands:

```
fix id all property/global youngsModulus peratomtype value_1 value_2 ...
    (value_i=value for Youngs Modulus of atom type i)
fix id all property/global poissonsRatio peratomtype value_1 value_2 ...
    (value_i=value for Poisson ratio of atom type i)
fix id all property/global coefficientRestitution peratomtypepair n_atomtypes value_11 value_12
    (value_ij=value for the coefficient of restitution between atom type i and j; n_atomtypes i
fix id all property/global coefficientFriction peratomtypepair n_atomtypes value_11 value_12 ..
    (value_ij=value for the (static) coefficient of friction between atom type i and j; n_atomt
```

IMPORTANT NOTE: You have to use atom styles beginning from 1, e.g. 1,2,3,...

This model contributes to surface heating in the frame of [surface sphere/heatable](#) if the appropriate flag is activated

Force Limiting:

Note, that not using limitForce might lead to attractive forces between particles and walls, especially in case the coefficient of restitution is small. Be sure you include this key word for the pair style and the wall model if you like to avoid this.

Restrictions:

If using SI units, youngsModulus must be > 5e6 If using CGS units, youngsModulus must be > 5e5 When using the limitForce, the specified coefficient of restitution is only approximate. This might become problematic for low coefficients of resitution as showin in Schwager and Poschel.

Default:

tangential_damping = 'on' *limitForce* = 'off' *heating_normal_hertz* = 'off'

(Di Renzo) Alberto Di Renzo, Francesco Paolo Di Maio, Chemical Engineering Science, 59 (3), p 525â 541 (2004).

(Ai) Jun Ai, Jian-Fei Chen, J. Michael Rotter, Jin Y. Ooi, Powder Technology, 206 (3), p 269-282 (2011).

(Brilliantov) Brilliantov, Spahn, Hertzsch, Poschel, Phys Rev E, 53, p 5382-5392 (1996). **(Schwager)** Schwager, Poschel, Gran Matt, 9, p 465-469 (2007).

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

(Zhang) Zhang and Makse, Phys Rev E, 72, p 011301 (2005).

gran model hertz/stiffness

Syntax:

```
model hertz [other model_type/model_name pairs as described here ] keyword values
```

• zero or more keyword/value pairs may be appended to the end (after all models are specified)

```
    limitForce values = 'on' or 'off'
        on = ensures that the normal force is never attractive (an artefact that can occur at the e
        off = standard implementation that might lead to attractive forces.
    tangential_damping values = 'on' or 'off'
        on = activates tangential damping
        off = no tangential damping
```

Description:

This granular model uses the following formula for the frictional force between two granular particles, when the distance r between two particles of radii Ri and Rj is less than their contact distance d = Ri + Rj. There is no force between the particles when r > d:

$$F = \left(k_n \underbrace{\delta n_{ij}}_{\substack{normal \\ overlap}} - \gamma_n \underbrace{vn_{ij}}_{\substack{normal \\ relative\ vel.}} \right) + \left(k_t \underbrace{\delta t_{ij}}_{\substack{tangential \\ overlap}} - \gamma_t \underbrace{vt_{ij}}_{\substack{tangential \\ relative\ v}} \right)$$

$$\underbrace{\qquad\qquad\qquad\qquad}_{normal\ force} \qquad \underbrace{\qquad\qquad\qquad\qquad}_{tangential\ force}$$

The tangential overlap is truncated to fulfil $F_t \leq X_\mu F_n$

In the first term is the normal force between the two particles and the second term is the tangential force. The normal force has 2 terms, a spring force and a damping force. The tangential force also has 2 terms: a shear force and a damping force. The shear force is a "history" effect that accounts for the tangential displacement ("tangential overlap") between the particles for the duration of the time they are in contact. This term is controlled by the tangential model in action Keyword *tangential_damping* can be used to eliminate the second part of the force in tangential direction. The way how the Coulomb friction limit acts is also controlled by the tangential model chosen by the user.

The quantities in the equations are as follows:

- delta_n = d - r = overlap distance of 2 particles
- k_n = elastic constant for normal contact
- k_t = elastic constant for tangential contact
- gamma_n = viscoelastic damping constant for normal contact

- gamma_t = viscoelastic damping constant for tangential contact
- delta_t = tangential displacement vector between 2 spherical particles which is truncated to satisfy a frictional yield criterion
- rmu = coefficient of rolling friction
- contactradius = contact radius, equal to particle radius - 0.5 * delta_n
- v_n = normal component of the relative velocity of the 2 particles
- v_t = tangential component of the relative velocity of the 2 particles
- w_r = relative rotational velocity of the 2 particles

For hertz/stiffness the coefficients used for the force calculation are non-linear, thus they are linked to the specified values by:

```
k_n = k_n_specified sqrt( delta_n R*)
k_t = k_t_specified sqrt( delta_n R*)
gamma_n = gamma_n_specified M* sqrt( delta_n R*)
gamma_t = gamma_t_specified M* sqrt( delta_n R*)
```

Where R* and M* are the effective radius and mass respectively.

To define those k_n_specified, k_t_specified, gamma_n_specified, and gamma_t_specified coefficients (material properties), it is mandatory to use multiple fix property/global commands:

```
fix id all property/global kn peratomtypepair n_atomtypes value_11 value_12 .. value_21 value_2
    (value_ij=value for k_n between atom type i and j; n_atomtypes is the number of atom types
fix id all property/global kt peratomtypepair n_atomtypes value_11 value_12 .. value_21 value_2
    (value_ij=value for k_t between atom type i and j; n_atomtypes is the number of atom types
fix id all property/global gamman peratomtypepair n_atomtypes value_11 value_12 .. value_21 val
    (value_ij=value for gamma_n between atom type i and j; n_atomtypes is the number of atom ty
fix id all property/global gammat peratomtypepair n_atomtypes value_11 value_12 .. value_21 val
    (value_ij=value for gamma_t between atom type i and j; n_atomtypes is the number of atom ty
```

IMPORTANT NOTE: You have to use atom styles beginning from 1, e.g. 1,2,3,...

Force Limiting:

Note, that not using limitForce might lead to attractive forces between particles and walls, especially in case the coefficient of restitution is small. Be sure you include this key word for the pair style and the wall model if you like to avoid this.

Restrictions:

If using SI units, youngsModulus must be > 5e6 If using CGS units, youngsModulus must be > 5e5 When using the limitForce, the specified coefficient of restitution is only approximate. This might become problematic for low coefficients of resitution as showin in Schwager and Poschel.

Default:

tangential_damping = 'on' *limitForce* = 'off'

(Di Renzo) Alberto Di Renzo, Francesco Paolo Di Maio, CES, 59 (3), p 525â 541 (2004).

(Ai) Jun Ai, Jian-Fei Chen, J. Michael Rotter, Jin Y. Ooi, Powder Technology, 206 (3), p 269-282 (2011).

(Brilliantov) Brilliantov, Spahn, Hertzsch, Poschel, Phys Rev E, 53, p 5382-5392 (1996).

(Schwager) Schwager, Poschel, Gran Matt, 9, p 465-469 (2007).

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

(Zhang) Zhang and Makse, Phys Rev E, 72, p 011301 (2005).

gran model hooke

Syntax:

```
model hooke [other model_type/model_name pairs as described here ] keyword values
```

• zero or more keyword/value pairs may be appended to the end (after all models are specified)

```
tangential_damping values = 'on' or 'off'
   on = activates tangential damping
   off = no tangential damping
 ktToKnUser values = 'on' or 'off'
   on = uses a different kt, namely kt = 2/7 * kn.
   off = standard implementation kt = kn.
 limitForce values = 'on' or 'off'
   on = ensures that the normal force is never attractive (an artefact that can occur at the e
   off = standard implementation that might lead to attractive forces.
 viscous = 'on' or 'off'
   on = restitution coefficient varies with a local Stokes number of the particle. Requires ad
   off = no modification to the restitution coefficient
 heating_normal_hooke values = 'on' or 'off'
   on = model contributes to surface heating in the frame of surface sphere/heatable
   off = model does not contributes to surface heating
```

Description:

This granular model uses the following formula for the frictional force between two granular particles,
when the distance r between two particles of radii Ri and Rj is less than their contact distance d = Ri + Rj.
There is no force between the particles when r > d:

$$F = (k_n \; \underbrace{\delta n_{ij}}_{\substack{normal \\ overlap}} - \gamma_n \; \underbrace{vn_{ij}}_{\substack{normal \\ relative\ vel.}}) + (k_t \; \underbrace{\delta t_{ij}}_{\substack{tangential \\ overlap}} - \gamma_t \; \underbrace{vt_{ij}}_{\substack{tangential \\ relative\ v}}$$

$$\underbrace{\hphantom{(k_n \; \delta n_{ij} - \gamma_n \; vn_{ij})}}_{normal\ force} \qquad \underbrace{\hphantom{(k_t \; \delta t_{ij} - \gamma_t \; vt_{ij})}}_{tangential\ force}$$

The tangential overlap is truncated to fulfil $F_t \leq X_\mu F_n$

In the first term is the normal force between the two particles and the second term is the tangential force.
The normal force has 2 terms, a spring force and a damping force. The tangential force also has 2 terms: a
shear force and a damping force. The shear force is a "history" effect that accounts for the tangential
displacement ("tangential overlap") between the particles for the duration of the time they are in contact.
This term is controlled by the tangential model in action

Keyword *tangential_damping* can be used to eliminate the second part of the force in tangential direction.

The quantities in the equations are as follows:

- delta_n = d - r = overlap distance of 2 particles
- k_n = elastic constant for normal contact
- k_t = elastic constant for tangential contact
- gamma_n = viscoelastic damping constant for normal contact
- gamma_t = viscoelastic damping constant for tangential contact
- delta_t = tangential displacement vector between 2 spherical particles which is truncated to satisfy a frictional yield criterion
- rmu = coefficient of rolling friction
- contactradius = contact radius, equal to particle radius - 0.5 * delta_n
- v_n = normal component of the relative velocity of the 2 particles
- v_t = tangential component of the relative velocity of the 2 particles
- w_r = relative rotational velocity of the 2 particles

The Kn, Kt, gamma_n, and gamma_t coefficients are calculated as follows from the material properties:

$$k_n = \frac{16}{15}\sqrt{R^* }Y^* \left(\frac{15 m^* V^2}{16\sqrt{R^*}\,Y^*} \right)^{1/5} ,$$

$$\gamma_n = \sqrt{ \frac{4 m^* k_n}{1 + \left(\dfrac{\pi}{ln(e)} \right)^2} } \geq 0,$$

$$k_t = k_n ,$$

$$\gamma_t = \gamma_n$$

$V...characteristic\ impact\ velocity$

gran model hooke

$$S_n = 2Y^* \sqrt{R^* \delta_n}, \qquad S_t = 8G^* \sqrt{R^* \delta_n}$$

$$\beta = \frac{ln(e)}{\sqrt{ln^2(e) + \pi^2}},$$

$$\frac{1}{Y^*} = \frac{\left(1 - v_1^2\right)}{Y_1} + \frac{\left(1 - v_2^2\right)}{Y_2},$$

$$\frac{1}{G^*} = \frac{2\left(2 - v_1\right)\left(1 + v_1\right)}{Y_1} + \frac{2\left(2 - v_2\right)\left(1 + v_2\right)}{Y_2}$$

$$\frac{1}{R^*} = \frac{1}{R_1} + \frac{1}{R_2}, \quad \frac{1}{m^*} = \frac{1}{m_1} + \frac{1}{m_2}$$

Y...Young's modulus   G...Shear modulus

v...Poisson ratio   e...coeff.of restitution

To define those material properties, it is mandatory to use multiple [fix property/global](#) commands:

```
fix id all property/global youngsModulus peratomtype value_1 value_2 ...
    (value_i=value for Youngs Modulus of atom type i)
fix id all property/global poissonsRatio peratomtype value_1 value_2 ...
    (value_i=value for Poisson ratio of atom type i)
fix id all property/global coefficientRestitution peratomtypepair n_atomtypes value_11 value_12
    (value_ij=value for the coefficient of restitution between atom type i and j; n_atomtypes i
fix id all property/global coefficientFriction peratomtypepair n_atomtypes value_11 value_12 ..
    (value_ij=value for the (static) coefficient of friction between atom type i and j; n_atomt
```

IMPORTANT NOTE: You have to use atom styles beginning from 1, e.g. 1,2,3,...

The "characteristic impact velocity" is additionally used for hooke:

```
fix id all property/global characteristicVelocity scalar value
    (value=value for characteristic impact velocity)
```

IMPORTANT NOTE: You have to use atom styles beginning from 1, e.g. 1,2,3,...

Force Limiting:

Note, that not using limitForce might lead to attractive forces between particles and walls, especially in case the coefficient of restitution is small. Be sure you include this key word for the pair style and the wall model if you like to avoid this.

Viscous model:

Using option *viscous* = stokes adapts the coefficient of restitution as proposed by [(Legendre)](#), *viscous* = off performs no modification.

One has to provide the 3 peratomtypepair parameters via a fix property/global command needed for the viscous damping:

```
fix id all property/global FluidViscosity peratomtypepair n_atomtypes value_11 value_12 .. valu
    (value_ij=value for fluid viscosity between atom type i and j; n_atomtypes is the number of
fix id all property/global CriticalStokes peratomtypepair n_atomtypes value_11 value_12 .. valu
    (value_ij=value for critical Stokes number between atom type i and j; n_atomtypes is the nu
fix id all property/global MaximumRestitution peratomtypepair n_atomtypes value_11 value_12 ..
    (value_ij=value for maximum coefficient of restitution between atom type i and j; n_atomtyp
```

IMPORTANT NOTE: You have to use atom styles beginning from 1, e.g. 1,2,3,...

This model contributes to surface heating in the frame of surface sphere/heatable if the appropriate flag is activated

Restrictions:

If using SI units, youngsModulus must be > 5e6 If using CGS units, youngsModulus must be > 5e5 When using *viscous*, FluidViscosity has to be > 0 When using the limitForce, the specified coefficient of restitution is only approximate. This might become problematic for low coefficients of resitution as showin in Schwager and Poschel.

Default:

viscous = 'off' *tangential_damping* = 'on' *ktToKnUser* = 'off' *limitForce* = 'off' *heating_normal_hooke* = 'off'

(Legendre) Legendre, Daniel and Guiraud. Phys. Fluids 17, 097105 (2005).

(Di Renzo) Alberto Di Renzo, Francesco Paolo Di Maio, Chemical Engineering Science, 59 (3), p 525â 541 (2004).

(Ai) Jun Ai, Jian-Fei Chen, J. Michael Rotter, Jin Y. Ooi, Powder Technology, 206 (3), p 269-282 (2011).

(Brilliantov) Brilliantov, Spahn, Hertzsch, Poschel, Phys Rev E, 53, p 5382-5392 (1996).

(Schwager) Schwager, Poschel, Gran Matt, 9, p 465-469 (2007).

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

(Zhang) Zhang and Makse, Phys Rev E, 72, p 011301 (2005).

gran model hooke/stiffness

Syntax:

```
model hooke/stiffness [other model_type/model_name pairs as described here ] keyword values
```

• zero or more keyword/value pairs may be appended to the end (after all models are specified)

```
absolute_damping values = 'on' or 'off'
    on = activates tangential damping
    off = no tangential damping
  limitForce values = 'on' or 'off'
    on = ensures that the normal force is never attractive (an artefact that can occur at the e
    off = standard implementation that might lead to attractive forces.
  tangential_damping values = 'on' or 'off'
    on = activates tangential damping
    off = no tangential damping
```

Description:

This granular model uses the following formula for the frictional force between two granular particles, when the distance r between two particles of radii Ri and Rj is less than their contact distance d = Ri + Rj. There is no force between the particles when r > d.

For the case of *absolute_damping* = 'off' (which is default), the specified damping coefficient is multiplied by the effective mass. The forces are implemented as

$$k_n = k_{n,specified},$$

$$\gamma_n = m_{eff}\, \gamma_{n,specified},$$

$$k_t = k_{t,specified},$$

$$\gamma_t = m_{eff}\, \gamma_{t,specified}.$$

For the case of *absolute_damping* = 'on', this multiplication is omitted and the forces become

$$k_n = k_{n,specified},$$

$$\gamma_n = \gamma_{n,specified},$$

$$k_t = k_{t,specified},$$

$$\gamma_t = \gamma_{t,specified}.$$

In the first term is the normal force between the two particles and the second term is the tangential force. The normal force has 2 terms, a spring force and a damping force. The tangential force also has 2 terms: a shear force and a damping force. The shear force is a "history" effect that accounts for the tangential displacement ("tangential overlap") between the particles for the duration of the time they are in contact. This term is controlled by the tangential model in action Keyword *tangential_damping* can be used to eliminate the second part of the force in tangential direction. The way how the Coulomb friction limit acts is also controlled by the tangential model chosen by the user.

The quantities in the equations are as follows:

- delta_n = d - r = overlap distance of 2 particles
- k_n = elastic constant for normal contact
- k_t = elastic constant for tangential contact
- gamma_n = viscoelastic damping constant for normal contact
- gamma_t = viscoelastic damping constant for tangential contact
- delta_t = tangential displacement vector between 2 spherical particles which is truncated to satisfy a frictional yield criterion
- rmu = coefficient of rolling friction
- contactradius = contact radius, equal to particle radius - 0.5 * delta_n
- v_n = normal component of the relative velocity of the 2 particles
- v_t = tangential component of the relative velocity of the 2 particles
- w_r = relative rotational velocity of the 2 particles

To define those k_n_specified, k_t_specified, gamma_n_specified, and gamma_t_specified coefficients (material properties), it is mandatory to use multiple fix property/global commands:

```
fix id all property/global kn peratomtypepair n_atomtypes value_11 value_12 .. value_21 value_2
    (value_ij=value for k_n between atom type i and j; n_atomtypes is the number of atom types
fix id all property/global kt peratomtypepair n_atomtypes value_11 value_12 .. value_21 value_2
    (value_ij=value for k_t between atom type i and j; n_atomtypes is the number of atom types
fix id all property/global gamman peratomtypepair n_atomtypes value_11 value_12 .. value_21 val
    (value_ij=value for gamma_n between atom type i and j; n_atomtypes is the number of atom ty
fix id all property/global gammat peratomtypepair n_atomtypes value_11 value_12 .. value_21 val
    (value_ij=value for gamma_t between atom type i and j; n_atomtypes is the number of atom ty
```

If the absolute damping implementation is used (*absolute_damping* = 'on'), the damping coefficients must be named gamman_abs and gammat_abs instead of gamman, gammat as follows:

```
fix id all property/global gamman_abs peratomtypepair n_atomtypes value_11 value_12 .. value_21
```

```
    (value_ij=value for gamma_n between atom type i and j; n_atomtypes is the number of atom ty
fix id all property/global gammat_abs peratomtypepair n_atomtypes value_11 value_12 .. value_21
    (value_ij=value for gamma_t between atom type i and j; n_atomtypes is the number of atom ty
```

IMPORTANT NOTE: You have to use atom styles beginning from 1, e.g. 1,2,3,...

Force Limiting:

Note, that not using limitForce might lead to attractive forces between particles and walls, especially in case the coefficient of restitution is small. Be sure you include this key word for the pair style and the wall model if you like to avoid this.

Restrictions:

If using SI units, youngsModulus must be > 5e6 If using CGS units, youngsModulus must be > 5e5 When using the limitForce, the specified coefficient of restitution is only approximate. This might become problematic for low coefficients of resitution as showin in Schwager and Poschel.

Default:

tangential_damping = 'on' *limitForce* = 'off'

(Di Renzo) Alberto Di Renzo, Francesco Paolo Di Maio, CES, 59 (3), p 525â 541 (2004).

(Ai) Jun Ai, Jian-Fei Chen, J. Michael Rotter, Jin Y. Ooi, Powder Technology, 206 (3), p 269-282 (2011).

(Brilliantov) Brilliantov, Spahn, Hertzsch, Poschel, Phys Rev E, 53, p 5382-5392 (1996).

(Schwager) Schwager, Poschel, Gran Matt, 9, p 465-469 (2007).

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

(Zhang) Zhang and Makse, Phys Rev E, 72, p 011301 (2005).

gran rolling_friction cdt

Syntax:

```
rolling_friction cdt
```

Description:

This model can be used as part of [pair gran](#) and [fix wall/gran](#)

The constant directional torque (CDT) model adds an additional torque contribution, equal to

torque_rf = rmu*k_n*delta_n*w_r_shear/mag(w_r_shear)*(R*).

w_r_shear is the projection of w_r into the shear plane, where w_r = w1-w2

If the rolling friction model is activated, the coefficient of rolling friction (rmu) must be defined as

```
fix id all property/global coefficientRollingFriction peratomtypepair n_atomtypes value_11 value_
    (value_ij=value for the coefficient of rolling friction between atom type i and j; n_atomtype
```

IMPORTANT NOTE: You have to use atom styles beginning from 1, e.g. 1,2,3,...

(Ai) Jun Ai, Jian-Fei Chen, J. Michael Rotter, Jin Y. Ooi, Powder Technology, 206 (3), p 269-282 (2011).

gran rolling_friction epsd2

Syntax:

```
rolling_friction epsd2
```

Description:

This model can be used as part of pair gran and fix wall/gran

The alternative elastic-plastic spring-dashpot (EPSD2) model (see Iwashita and Oda) adds an additional torque contribution. It is similar to the EPSD model, but in contrast to the original model the rolling stiffness k_r is defined as

$$k_r = k_t \cdot R^{*2}$$

where k_t is the abovementioned tangential stiffness. Furthermore, the viscous damping torque M_rd is disabled at all.

The coefficient of rolling friction (rmu) must be defined as

```
fix id all property/global coefficientRollingFriction peratomtypepair n_atomtypes value_11 value_
    (value_ij=value for the coefficient of rolling friction between atom type i and j; n_atomtype
```

```
This coefficient rmu is equal to the rmu as defined in the CDT model.
```

IMPORTANT NOTE: You have to use atom styles beginning from 1, e.g. 1,2,3,...

(Ai) Jun Ai, Jian-Fei Chen, J. Michael Rotter, Jin Y. Ooi, Powder Technology, 206 (3), p 269-282 (2011).

gran rolling_friction epsd3

Syntax:

```
rolling_friction epsd3
```

Description:

This model can be used as part of pair gran and fix wall/gran

The elastic-plastic spring-dashpot (EPSD) model (see Ai et al.) adds an additional torque contribution, equal
to

$$M_{\mathrm{r}} = M_{\mathrm{r}}^{\mathrm{k}} + M_{\mathrm{r}}^{\mathrm{d}}$$

where the torque due to the spring M_rk is calculated as

$$k_{\mathrm{r}} = 2.25 k_{\mathrm{n}} \mu_{\mathrm{r}}^2 R^{*2}$$
$$\Delta M_{\mathrm{r}}^{\mathrm{k}} = -k_{\mathrm{r}} \Delta \theta_{\mathrm{r}}$$
$$M_{\mathrm{r,t}+\Delta \mathrm{t}}^{\mathrm{k}} = M_{\mathrm{r,t}}^{\mathrm{k}} + \Delta M_{\mathrm{r}}^{\mathrm{k}}$$
$$|M_{\mathrm{r,t}+\Delta \mathrm{t}}^{\mathrm{k}}| \leq M_{\mathrm{r}}^{\mathrm{m}}$$
$$M_{\mathrm{r}}^{\mathrm{m}} = \mu_{\mathrm{r}} R^* F_{\mathrm{n}}$$

Here k_r denotes the rolling stiffness and dtheta_r is the incremental relative rotation between the particles.
The spring torque is limited by the full mobilisation torque M_rm that is determined by the normal force F_n
and the coefficient of rolling friction (rmu) (compare the CDT model).

The rolling stiffness k_r is computed using the "coeffRollingStiffness" prefactor, that needs to be defined by
the user (see below). k_r is computed from k_r = coeffRollingStiffness*k_n*rmu*rmu*reff*reff.

The viscous damping torque M_rd is implemented as

$$M_{\mathrm{r,t}+\Delta \mathrm{t}}^{\mathrm{d}} = \begin{cases} -C_{\mathrm{r}} \dot{\theta}_{\mathrm{r}} & \text{if} |M_{\mathrm{r,t}+\Delta \mathrm{t}}^{\mathrm{k}}| < M_{\mathrm{r}}^{\mathrm{m}} \\ -f C_{\mathrm{r}} \dot{\theta}_{\mathrm{r}} & \text{if} |M_{\mathrm{r,t}+\Delta \mathrm{t}}^{\mathrm{k}}| = M_{\mathrm{r}}^{\mathrm{m}} \end{cases}$$

where in the current implementation the damping is disabled in case of full mobilisation (f = 0). The damping
coefficient C_r may be expressed as:

$$C_r = \eta_r C_r^{\text{crit}}$$

$$C_r^{\text{crit}} = 2\sqrt{I_r k_r}$$

$$I_r = \left(\frac{1}{I_i + m_i r_i^2} + \frac{1}{I_j + m_j r_j^2} \right)^{-1}$$

Here $I_{i/j}$ is the moment of inertia and $m_{i/j}$ is the mass of the particles i and j, respectively.

The coefficient of rolling friction (rmu) must be defined as

```
fix id all property/global coefficientRollingFriction peratomtypepair n_atomtypes value_11 value_
    (value_ij=value for the coefficient of rolling friction between atom type i and j; n_atomtype
```

This coefficient rmu is equal to the rmu as defined in the CDT model. In addition to rmu, eta_r is the required material property that must be defined as

```
fix id all property/global coefficientRollingViscousDamping peratomtypepair n_atomtypes value_11
    (value_ij=value for the coefficient of rolling friction between atom type i and j; n_atomtype
```

The "coeffRollingStiffness" prefactor needs to be defined by the user as

```
fix     id all property/global coeffRollingStiffness scalar value
```

Please see Ai et al., 2011 for discussion.

IMPORTANT NOTE: You have to use atom styles beginning from 1, e.g. 1,2,3,...

(Ai) Jun Ai, Jian-Fei Chen, J. Michael Rotter, Jin Y. Ooi, Powder Technology, 206 (3), p 269-282 (2011).

gran rolling_friction epsd

Syntax:

```
rolling_friction epsd
```

Description:

This model can be used as part of pair gran and fix wall/gran

The elastic-plastic spring-dashpot (EPSD) model (see Ai et al.) adds an additional torque contribution, equal to

$$M_{\mathrm{r}} = M_{\mathrm{r}}^{\mathrm{k}} + M_{\mathrm{r}}^{\mathrm{d}}$$

where the torque due to the spring M_rk is calculated as

$$k_{\mathrm{r}} = 2.25 k_{\mathrm{n}} \mu_{\mathrm{r}}^2 R^{*2}$$
$$\Delta M_{\mathrm{r}}^{\mathrm{k}} = -k_{\mathrm{r}} \Delta\theta_{\mathrm{r}}$$
$$M_{\mathrm{r},\mathrm{t}+\Delta\mathrm{t}}^{\mathrm{k}} = M_{\mathrm{r},\mathrm{t}}^{\mathrm{k}} + \Delta M_{\mathrm{r}}^{\mathrm{k}}$$
$$|M_{\mathrm{r},\mathrm{t}+\Delta\mathrm{t}}^{\mathrm{k}}| \leq M_{\mathrm{r}}^{\mathrm{m}}$$
$$M_{\mathrm{r}}^{\mathrm{m}} = \mu_{\mathrm{r}} R^* F_{\mathrm{n}}$$

Here k_r denotes the rolling stiffness that depends on the stiffness of the normal spring (from the normal contact law), the effective radius and the coefficient of rolling friction (mu_r). Following **(Ai)** the prefactor of 2.25 is valid for 3D simulations. The EPSD3 model allows to modify the prefactor. dtheta_r is the incremental relative rotation between the particles. The spring torque is limited by the full mobilisation torque M_rm that is determined by the normal force F_n and the coefficient of rolling friction (mu_r) (compare the CDT model).

The viscous damping torque M_rd is implemented as

$$M_{\mathrm{r},\mathrm{t}+\Delta\mathrm{t}}^{\mathrm{d}} = \begin{cases} -C_{\mathrm{r}}\dot{\theta}_{\mathrm{r}} & \text{if} |M_{\mathrm{r},\mathrm{t}+\Delta\mathrm{t}}^{\mathrm{k}}| < M_{\mathrm{r}}^{\mathrm{m}} \\ -fC_{\mathrm{r}}\dot{\theta}_{\mathrm{r}} & \text{if} |M_{\mathrm{r},\mathrm{t}+\Delta\mathrm{t}}^{\mathrm{k}}| = M_{\mathrm{r}}^{\mathrm{m}} \end{cases}$$

where in the current implementation the damping is disabled in case of full mobilisation (f = 0). The damping coefficient C_r may be expressed as:

$$C_{\mathrm{r}} = \eta_{\mathrm{r}} C_{\mathrm{r}}^{\mathrm{crit}}$$

$$C_{\mathrm{r}}^{\mathrm{crit}} = 2\sqrt{I_{\mathrm{r}} k_{\mathrm{r}}}$$

$$I_{\mathrm{r}} = \left(\frac{1}{I_i + m_i r_i^2} + \frac{1}{I_j + m_j r_j^2} \right)^{-1}$$

Here I_i/j is the moment of inertia and m_i/j is the mass of the particles i and j, respectively.

The coefficient of rolling friction (mu_r) must be defined as

```
fix id all property/global coefficientRollingFriction peratomtypepair n_atomtypes value_11 value_
      (value_ij=value for the coefficient of rolling friction between atom type i and j; n_atomtype
```

This coefficient mu_r is equal to the mu_r as defined in the CDT model. In addition to mu_r, eta_r is the required material property that must be defined as

```
fix id all property/global coefficientRollingViscousDamping peratomtypepair n_atomtypes value_11
      (value_ij=value for the coefficient of rolling friction between atom type i and j; n_atomtype
```

IMPORTANT NOTE: You have to use atom styles beginning from 1, e.g. 1,2,3,...

(**Ai**) Jun Ai, Jian-Fei Chen, J. Michael Rotter, Jin Y. Ooi, Powder Technology, 206 (3), p 269-282 (2011).

gran surface sphere

Syntax:

```
surface sphere
```

Description:

This is the default surface model for a particle, it prepresents a smooth (non-rough) sphere. It is the default surface model for all LIGGGHTS(R)-WSU simulations. In case of a multi-sphere simulation, this surface model is applied to all spheres within one rigid body.

Restrictions: none

gran tangential history

Syntax:

```
tangential history [other model_type/model_name pairs as described here ] keyword values
```

- zero or more keyword/value pairs may be appended to the end (after all models are specified)

```
heating_tangential_history values = 'on' or 'off'
    on = model contributes to surface heating in the frame of surface sphere/heatable
    off = model does not contributes to surface heating
```

Description:

This granular model is based on the general description of granular force interaction as described in [pair gran](#).

The spring part of the tangential force (k_t) is a "history" effect that accounts for the tangential displacement ("tangential overlap") between the particles for the duration of the time they are in contact. It is calculated by adding up the releative tangential velocity at the contact point times the time-step size.

If this model is chosen, then this "tangential overlap" spring force is actually calculated / taken into account.

The coefficient of friction cof is the upper limit of the tangential force through the Coulomb criterion Ft_spring = cof*Fn, where Ft_spring and Fn are the tangential spring and total normal force components. Thus in the Hookean case, the tangential force between 2 particles grows according to a tangential spring and dash-pot model until Ft_spring/Fn = cof and is then held at Ft_spring = Fn*cof until the particles lose contact. In the Hertzian case and other non-linear cases, a similar analogy holds, though the spring is no longer linear.

The damping contribution is only added in time-steps where there is no slip, i.e. the Coulomb criterion is not met.

This model contributes to surface heating in the frame of [surface sphere/heatable](#) if the appropriate flag is activated

Default:

heating_tangential_history = 'off'

gran tangential no_history

Syntax:

```
tangential no_history
```

LIGGGHTS(R)-WSU vs. LIGGGHTS(R)-WSU Info:

This part of [pair gran](#) and [fix wall/gran](#) is not availabe in LIGGGHTS(R)-WSU.

Description:

This granular model is based on the general description of granular force interaction as described in [pair gran](#).

If this model is chose, then this "tangential overlap" spring force is NOT calculated / taken into account, i.e. $k_t = 0$.

The coefficient of friction cof is the upper limit of the tangential force through the Coulomb criterion Ft = cof*Fn, where Ft and Fn are the tangential spring and normal force components in the formulas above.

group command

Syntax:

```
group ID style args
```

- ID = user-defined name of the group
- style = *delete* or *region* or *type* or *id* or *molecule* or *variable* or *subtract* or *union* or *intersect*

```
  delete = no args
    region args = region-ID
    type or id or molecule
      args = list of one or more atom types, atom IDs, or molecule IDs
        any entry in list can be a sequence formatted as A:B or A:B:C where
        A = starting index, B = ending index,
        C = increment between indices, 1 if not specified
      args = logical value
        logical = "" or ">=" or "==" or "!="
        value = an atom type or atom ID or molecule ID (depending on style)
      args = logical value1 value2
        logical = ""
        value1,value2 = atom types or atom IDs or molecule IDs (depending on style)
    variable args  = variable-ID
    subtract args = two or more group IDs
    union args = one or more group IDs
    intersect args = two or more group IDs
```

Examples:

```
group edge region regstrip
group water type 3 4
group sub id 10 25 50
group sub id 10 25 50 500:1000
group sub id 100:10000:10
group sub id <= 150
group polyA molecule  50 250
group hienergy variable eng
group boundary subtract all a2 a3
group boundary union lower upper
group boundary intersect upper flow
group boundary delete
```

Description:

Identify a collection of atoms as belonging to a group. The group ID can then be used in other commands such as fix, compute, dump, or velocity to act on those atoms together.

If the group ID already exists, the group command adds the specified atoms to the group.

The *delete* style removes the named group and un-assigns all atoms that were assigned to that group. Since there is a restriction (see below) that no more than 32 groups can be defined at any time, the *delete* style allows you to remove groups that are no longer needed, so that more can be specified. You cannot delete a group if it has been used to define a current fix or compute or dump.

The *region* style puts all atoms in the region volume into the group. Note that this is a static one-time assignment. The atoms remain assigned (or not assigned) to the group even in they later move out of the region volume.

group command   346

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 use arguments specified in one of two formats.

The first 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*. Each entry in the list can be a colon-separated sequence A:B or A:B:C, as in two of the examples above. A "sequence" generates a sequence of values (types or IDs), with an optional increment. The first example with 500:1000 has the default increment of 1 and would add all atom IDs from 500 to 1000 (inclusive) to the group sub, along with 10,25,50 since they also appear in the list of values. The second example with 100:10000:10 uses an increment of 10 and would thus would add atoms IDs 100,110,120, ... 9990,10000 to the group sub.

The second format is a *logical* followed by one or two values (type or ID). The 7 valid logicals are listed above. All the logicals except take a single argument. The 3rd example above adds all atoms with IDs from 1 to 150 to the group named *sub*. The logical means "between" and takes 2 arguments. The 4th example above adds all atoms belonging to molecules with IDs from 50 to 250 (inclusive) to the group named polyA.

The *variable* style evaluates a variable to determine which atoms to add to the group. It must be an [atom-style variable](#) previously defined in the input script. If the variable evaluates to a non-zero value for a particular atom, then that atom is added to the specified group.

Atom-style variables can specify formulas that include thermodynamic quantities, per-atom values such as atom coordinates, or per-atom quantities calculated by computes, fixes, or other variables. They can also include Boolean logic where 2 numeric values are compared to yield a 1 or 0 (effectively a true or false). Thus using the *variable* style, is a general way to flag specific atoms to include or exclude from a group.

For example, these lines define a variable "eatom" that calculates the potential energy of each atom and includes it in the group if its potential energy is above the threshhold value -3.0.

```
compute         1 all pe/atom
compute         2 all reduce sum c_1
thermo_style    custom step temp pe c_2
run             0

variable        eatom atom "c_1 > -3.0"
group           hienergy variable eatom
```

Note that these lines

```
compute         2 all reduce sum c_1
thermo_style    custom step temp pe c_2
run             0
```

are necessary to insure that the "eatom" variable is current when the group command invokes it. Because the eatom variable computes the per-atom energy via the pe/atom compute, it will only be current if a run has been performed which evaluated pairwise energies, and the pe/atom compute was actually invoked during the run. Printing the thermodyanmic info for compute 2 insures that this is the case, since it sums the pe/atom compute values (in the reduce compute) to output them to the screen. See the "Variable Accuracy" section of the [variable](#) doc page for more details on insuring that variables are current when they are evaluated between runs.

The *subtract* style takes a list of two or more existing group names as arguments. All atoms that belong to the 1st group, but not to any of the other groups are added to the specified group.

The *union* style takes a list of one or more existing group names as arguments. All atoms that belong to any of the listed groups are added to the specified group.

The *intersect* style takes a list of two or more existing group names as arguments. Atoms that belong to every one of the listed groups are added to the specified group.

A group with the ID *all* is predefined. All atoms belong to this group. This group cannot be deleted.

Restrictions:

There can be no more than 32 groups defined at one time, including "all".

Related commands:

[dump](), [fix](), [region](), [velocity]()

Default:

All atoms belong to the "all" group.

if command

Syntax:

```
if boolean then t1 t2 ... elif boolean f1 f2 ... elif boolean f1 f2 ... else e1 e2 ...
```

- boolean = a Boolean expression evaluated as TRUE or FALSE (see below)
- then = required word
- t1,t2,...,tN = one or more LIGGGHTS(R)-WSU commands to execute if condition is met, each enclosed in quotes
- elif = optional word, can appear multiple times
- f1,f2,...,fN = one or more LIGGGHTS(R)-WSU commands to execute if elif condition is met, each enclosed in quotes (optional arguments)
- else = optional argument
- e1,e2,...,eN = one or more LIGGGHTS(R)-WSU commands to execute if no condition is met, each enclosed in quotes (optional arguments)

Examples:

```
if "${steps} > 1000" then quit
if "$x <= $y" then "print X is smaller = $x" else "print Y is smaller = $y"
if "(${eng} > 0.0) || ($n <1000)" then &
  "timestep 0.005" &
elif $n ${eng_previous}" then "jump file1" else "jump file2"
```

Description:

This command provides an if-then-else capability within an input script. A Boolean expression is evaluted and the result is TRUE or FALSE. Note that as in the examples above, the expression can contain variables, as defined by the variable command, which will be evaluated as part of the expression. Thus a user-defined formula that reflects the current state of the simulation can be used to issue one or more new commands.

If the result of the Boolean expression is TRUE, then one or more commands (t1, t2, ..., tN) are executed. If it is FALSE, then Boolean expressions associated with successive elif keywords are evaluated until one is found to be true, in which case its commands (f1, f2, ..., fN) are executed. If no Boolean expression is TRUE, then the commands associated with the else keyword, namely (e1, e2, ..., eN), are executed. The elif and else keywords and their associated commands are optional. If they aren't specified and the initial Boolean expression is FALSE, then no commands are executed.

The syntax for Boolean expressions is described below.

Each command (t1, f1, e1, etc) can be any valid LIGGGHTS(R)-WSU input script command, except an include command, which is not allowed. If the command is more than one word, it must enclosed in quotes, so it will be treated as a single argument, as in the examples above.

IMPORTANT NOTE: If a command itself requires a quoted argument (e.g. a print command), then double and single quotes can be used and nested in the usual manner, as in the examples above and below. See Section_commands 2 of the manual for more details on using quotes in arguments. Only one of level of nesting is allowed, but that should be sufficient for most use cases.

Note that by using the line continuation character "&", the if command can be spread across many lines, though it is still a single command:

```
if "$a <$b" then &
  "print 'Minimum value = $a'" &
  "run 1000" &
else &
  'print "Minimum value = $b"' &
  "minimize 0.001 0.001 1000 10000"
```

Note that if one of the commands to execute is quit (of an invalid LIGGGHTS(R)-WSU command such as "blah"), as in the first example above, then executing the command will cause LIGGGHTS(R)-WSU to halt.

Note that by jumping to a label in the same input script, the if command can be used to break out of a loop. See the variable delete command for info on how to delete the associated loop variable, so that it can be re-used later in the input script.

Here is an example of a double loop which uses the if and jump commands to break out of the inner loop when a condition is met, then continues iterating thru the outer loop.

```
label       loopa
variable    a loop 5
  label       loopb
  variable  b loop 5
  print     "A,B = $a,$b"
  run       10000
  if        '$b > 2' then "print 'Jumping to another script'" "jump in.script break"
  next      b
  jump      in.script loopb
label       break
variable    b delete

next        a
jump        in.script loopa
```

The Boolean expressions for the if and elif keywords have a C-like syntax. Note that each expression is a single argument within the if command. Thus if you want to include spaces in the expression for clarity, you must enclose the entire expression in quotes.

An expression is built out of numbers:

```
0.2, 100, 1.0e20, -15.4, etc
```

and Boolean operators:

```
A == B, A != B, A <B, A  <= B, A > B, A >= B, A && B, A || B, !A
```

Each A and B is a number or a variable reference like $a or ${abc}, or another Boolean expression.

If a variable is used it must produce a number when evaluated and substituted for in the expression, else an error will be generated.

Expressions are evaluated left to right and have the usual C-style precedence: the unary logical NOT operator "!" has the highest precedence, the 4 relational operators "", and ">=" are next; the two remaining relational operators "==" and "!=" are next; then the logical AND operator "&&"; and finally the logical OR operator "||" has the lowest precedence. Parenthesis can be used to group one or more portions of an expression and/or enforce a different order of evaluation than what would occur with the default precedence.

The 6 relational operators return either a 1.0 or 0.0 depending on whether the relationship between x and y is TRUE or FALSE. The logical AND operator will return 1.0 if both its arguments are non-zero, else it returns 0.0. The logical OR operator will return 1.0 if either of its arguments is non-zero, else it returns 0.0. The

logical NOT operator returns 1.0 if its argument is 0.0, else it returns 0.0.

The overall Boolean expression produces a TRUE result if the result is non-zero. If the result is zero, the expression result is FALSE.

Restrictions: none

Related commands:

variable, print

Default: none

include command

Syntax:

```
include file
```

- file = filename of new input script to switch to

Examples:

```
include newfile
include in.run2
```

Description:

This command opens a new input script file and begins reading LIGGGHTS(R)-WSU commands from that file. When the new file is finished, the original file is returned to. Include files can be nested as deeply as desired. If input script A includes script B, and B includes A, then LIGGGHTS(R)-WSU could run for a long time.

If the filename is a variable (see the [variable](#) command), different processor partitions can run different input scripts.

Restrictions: none

Related commands:

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

Default: none

Compile LIGGGHTS(R)-WSU for Windows

Description:

This routine describes how to setup your system in order to compile LIGGGHTS(R)-WSU for Windows.

Prerequisites:

All mentioned programs are available for free.

MPI:

LIGGGHTS(R)-WSU is a highly parallelized simulation engine. If you want to run simulations in parallel you have to install MPI (Message Passing Interface standard). We suggest to use the implementation of Microsoft, which is available here. You need the binaries and the header files (included in the SDK). Please follow the instructions of the installer.

Git:

GIT is an open-source version control system. We provide LIGGGHTS(R)-WSU via github. It is to your advantage to use any git program to keep your LIGGGHTS(R)-WSU version up-to-date. We recommend to use git-scm as it provides a minimum bash environment too.

You can set up your GIT environment to use the SSH-Keys thus you don't have to enter your user and password multiple times. Check the GIT GUI documentation for more details.

Important: Git-scm will ask during installation how it should handle newlines; please use "Checkout as-is, commit Unix-style line endings".

Python or Cygwin:

A python script updates the VS-project, thus you have to install either python or cygwin. In case of cygwin you can use another GIT program, because cygwin provides also a bash environment. The description of the procedure assumes that you have installed cygwin.

Visual studio:

To compile LIGGGHTS(R)-WSU you need a development environment. You can download Microsoft Visual Studio Express (current version: Express 2013 with Update 4 for Windows Desktop) for free from www.visualstudio.com. Therefore, you have to create an Microsoft account.

VTK support:

A detailed description how to compile LIGGGHTS(R)-WSU with VTK support will follow here.

Procedure:

Basically the following steps have to be performed:

- download/clone your repository from github
- update auto-generated header-files
- update your VS-project
- compile your LIGGGHTS(R)-WSU version

Download/clone your LIGGGHTS(R)-WSU version:

In this tutorial we use the suggested [git-scm](). You can use either the *Git GUI* or the *Git Bash*. In case of a bash you can follow the instructions in the documentation [githubAccess_non-public](). Otherwise copy the link to your repository, for instance *https://github.com/CFDEMproject/LIGGGHTS-COMPANY.git*, into the *Git GUI* and save the repository at your computer, e.g. *C:/repositories/LIGGGHTS-COMPANY*.

In the directory *LIGGGHTS-COMPANY/src/WINDOWS* you find an README, which describes the following steps in detail.

Update auto-generated header-files:

Start *cygwin* and change your current directory to the src-directory inside of your repository, for instance

```
cd /cygdrive/c/repositories/LIGGGHTS-COMPANY/src
```

Update the header files by

```
sh Make.sh styles
sh Make.sh models
```

You can check the files by

```
ls style_*
```

which should output a list of style file headers.

Update your VS-project:

In order to update the Visual Studio project in die WINDOWS directory run following commands (still in cygwin):

```
cd WINDOWS
python update_project.py LIGGGHTS.vcxproj
```

Compile your LIGGGHTS(R)-WSU version:

Finally to compile your LIGGGHTS(R)-WSU version open the file *LIGGGHTS_VS2013.sln* with Visual Studio.

To compile with MPI:

1. Switch to the correct setting for your system. *Debug* or *Release* and *32bit* or *64bit*.
2. Check if the MPI include and lib directories are correctly set in the project properties of LIGGGHTS(R)-WSU - Open the *LIGGGHTS/Properties* and go to *Configuration Properties/VC++ Directories*. For instance, a common include path is *C:\Program Files x86\Microsoft SDKs\MPI\Include*.
3. Build the LIGGGHTS(R)-WSU project.

To compile without MPI:

1. Build the STUBS project.
2. Build LIGGGHTS(R)-WSU using Debug_STUBS or Release_STUBS configurations from the provided project. (use x64 for 64bit binary)

You have now generated the executable **LIGGGHTS.exe** in the sub-directory *LIGGGHTS-COMPANY\src\WINDOWS\Release*. You can add this directory to your PATH environment variable to shorten the call command.

Run an example

You can start a LIGGGHTS(R)-WSU simulation inside of *cygwin* or a *command prompt* (search for cmd.exe). Assuming you use your cygwin, change to one example in your repository.

```
cd /cygdrive/c/repositories/LIGGGHTS-COMPANY/examples/LIGGGHTS/Tutorials_public/chute_wear
```

and start the simulation by

```
/cygdrive/c/repositories/LIGGGHTS-COMPANY/src/WINDOWS/Release/LIGGGHTS.exe -in in.chute_wear
```

If you added the Release-folder to your PATH variable the command shortens to

```
LIGGGHTS.exe -in in.chute_wear
```

Questions?

If any questions remain, contact us.

jump command

Syntax:

```
jump file label
```

- file = filename of new input script to switch to
- label = optional label within file to jump to

Examples:

```
jump newfile
jump in.run2 runloop
jump SELF runloop
```

Description:

This command closes the current input script file, opens the file with the specified name, and begins reading LIGGGHTS(R)-WSU commands from that file. Unlike the include command, the original file is not returned to, although by using multiple jump commands it is possible to chain from file to file or back to the original file.

If the word "SELF" is used for the filename, then the current input script is re-opened and read again.

IMPORTANT NOTE: The SELF option is not guaranteed to work when the current input script is being read through stdin (standard input), e.g.

```
lmp_g++ <in.script
```

since the SELF option invokes the C-library rewind() call, which may not be supported for stdin on some systems. This can be worked around by using the -in command-line argument or the -var command-line argument to pass the script name as a variable to the input script In the latter case, the "fname" variable could be used in place of SELF. E.g.

```
lmp_g++ -in in.script
```

```
lmp_g++ -var fname n.script <in.script
```

The 2nd argument to the jump command is optional. If specified, it is treated as a label and the new file is scanned (without executing commands) until the label is found, and commands are executed from that point forward. This can be used to loop over a portion of the input script, as in this example. These commands perform 10 runs, each of 10000 steps, and create 10 dump files named file.1, file.2, etc. The next command is used to exit the loop after 10 iterations. When the "a" variable has been incremented for the tenth time, it will cause the next jump command to be skipped.

```
variable a loop 10
label loop
dump 1 all atom 100 file.$a
run 10000
undump 1
next a
jump in.lj loop
```

If the jump *file* argument is a variable, the jump command can be used to cause different processor partitions to run different input scripts. In this example, LIGGGHTS(R)-WSU is run on 40 processors, with 4 partitions of 10 procs each. An in.file containing the example variable and jump command will cause each partition to run a different simulation.

```
mpirun -np 40 lmp_ibm -partition 4x10 -in in.file


variable f world script.1 script.2 script.3 script.4
jump $f
```

Here is an example of a double loop which uses the [if](#) and jump commands to break out of the inner loop when a condition is met, then continues iterating thru the outer loop.

```
label      loopa
variable   a loop 5
  label     loopb
  variable  b loop 5
  print     "A,B = $a,$b"
  run       10000
  if        $b > 2 then "jump in.script break"
  next      b
  jump      in.script loopb
label      break
variable   b delete

next       a
jump       in.script loopa
```

Restrictions:

If you jump to a file and it does not contain the specified label, LIGGGHTS(R)-WSU will come to the end of the file and exit.

Related commands:

[variable](#), [include](#), [label](#), [next](#)

Default: none

label command

Syntax:

```
label ID
```

- ID = string used as label name

Examples:

```
label xyz
label loop
```

Description:

Label this line of the input script with the chosen ID. Unless a jump command was used previously, this does nothing. But if a [jump](#) command was used with a label argument to begin invoking this script file, then all command lines in the script prior to this line will be ignored. I.e. execution of the script will begin at this line. This is useful for looping over a section of the input script as discussed in the [jump](#) command.

Restrictions: none

Related commands: none

Default: none

lattice command

Syntax:

```
lattice style scale keyword values ...
```

- style = *none* or *sc* or *bcc* or *fcc* or *hcp* or *diamond* or *sq* or *sq2* or *hex* or *custom*
- scale = scale factor between lattice and simulation box

```
    scale = reduced density rho* (for LJ units)
    scale = lattice constant in distance units (for all other units)
```
- zero or more keyword/value pairs may be appended
- keyword = *origin* or *orient* or *spacing* or *a1* or *a2* or *a3* or *basis*

```
  origin values = x y z
     x,y,z = fractions of a unit cell (0 <= x,y,z <1)
  orient values = dim i j k
    dim = x or y or z
    i,j,k = integer lattice directions
  spacing values = dx dy dz
    dx,dy,dz = lattice spacings in the x,y,z box directions
  a1,a2,a3 values = x y z
    x,y,z = primitive vector components that define unit cell
  basis values = x y z
    x,y,z = fractional coords of a basis atom (0  <= x,y,z <1)
```

Examples:

```
lattice fcc 3.52
lattice hex 0.85
lattice sq 0.8 origin 0.0 0.5 0.0 orient x 1 1 0 orient y -1 1 0
lattice custom 3.52 a1 1.0 0.0 0.0 a2 0.5 1.0 0.0 a3 0.0 0.0 0.5 &
                basis 0.0 0.0 0.0 basis 0.5 0.5 0.5
lattice none 2.0
```

Description:

Define a lattice for use by other commands. In LIGGGHTS(R)-WSU, a lattice is simply a set of points in space, determined by a unit cell with basis atoms, that is replicated infinitely in all dimensions. The arguments of the lattice command can be used to define a wide variety of crystallographic lattices.

A lattice is used by LIGGGHTS(R)-WSU in two ways. First, the [create_atoms](#) command creates atoms on the lattice points inside the simulation box. Note that the [create_atoms](#) command allows different atom types to be assigned to different basis atoms of the lattice. Second, the lattice spacing in the x,y,z dimensions implied by the lattice, can be used by other commands as distance units (e.g. [create_box](#), [region](#) and [velocity](#)), which are often convenient to use when the underlying problem geometry is atoms on a lattice.

The lattice style must be consistent with the dimension of the simulation - see the [dimension](#) command. Styles *sc* or *bcc* or *fcc* or *hcp* or *diamond* are for 3d problems. Styles *sq* or *sq2* or *hex* are for 2d problems. Style *custom* can be used for either 2d or 3d problems.

A lattice consists of a unit cell, a set of basis atoms within that cell, and a set of transformation parameters (scale, origin, orient) that map the unit cell into the simulation box. The vectors a1,a2,a3 are the edge vectors of the unit cell. This is the nomenclature for "primitive" vectors in solid-state crystallography, but in LIGGGHTS(R)-WSU the unit cell they determine does not have to be a "primitive cell" of minimum volume.

A lattice of style *none* does not define a unit cell and basis set, so it cannot be used with the create_atoms command. However it does define a lattice spacing via the specified scale parameter. As explained above the lattice spacings in x,y,z can be used by other commands as distance units. No additional keyword/value pairs can be specified for the *none* style. By default, a "lattice none 1.0" is defined, which means the lattice spacing is the same as one distance unit, as defined by the units command.

Lattices of style *sc*, *fcc*, *bcc*, and *diamond* are 3d lattices that define a cubic unit cell with edge length = 1.0. This means a1 = 1 0 0, a2 = 0 1 0, and a3 = 0 0 1. Style *hcp* has a1 = 1 0 0, a2 = 0 sqrt(3) 0, and a3 = 0 0 sqrt(8/3). The placement of the basis atoms within the unit cell are described in any solid-state physics text. A *sc* lattice has 1 basis atom at the lower-left-bottom corner of the cube. A *bcc* lattice has 2 basis atoms, one at the corner and one at the center of the cube. A *fcc* lattice has 4 basis atoms, one at the corner and 3 at the cube face centers. A *hcp* lattice has 4 basis atoms, two in the z = 0 plane and 2 in the z = 0.5 plane. A *diamond* lattice has 8 basis atoms.

Lattices of style *sq* and *sq2* are 2d lattices that define a square unit cell with edge length = 1.0. This means a1 = 1 0 0 and a2 = 0 1 0. A *sq* lattice has 1 basis atom at the lower-left corner of the square. A *sq2* lattice has 2 basis atoms, one at the corner and one at the center of the square. A *hex* style is also a 2d lattice, but the unit cell is rectangular, with a1 = 1 0 0 and a2 = 0 sqrt(3) 0. It has 2 basis atoms, one at the corner and one at the center of the rectangle.

A lattice of style *custom* allows you to specify a1, a2, a3, and a list of basis atoms to put in the unit cell. By default, a1 and a2 and a3 are 3 orthogonal unit vectors (edges of a unit cube). But you can specify them to be of any length and non-orthogonal to each other, so that they describe a tilted parallelepiped. Via the *basis* keyword you add atoms, one at a time, to the unit cell. Its arguments are fractional coordinates (0.0 <= x,y,z < 1.0), so that a value of 0.5 means a position half-way across the unit cell in that dimension.

This sub-section discusses the arguments that determine how the idealized unit cell is transformed into a lattice of points within the simulation box.

The *scale* argument determines how the size of the unit cell will be scaled when mapping it into the simulation box. I.e. it determines a multiplicative factor to apply to the unit cell, to convert it to a lattice of the desired size and distance units in the simulation box. The meaning of the *scale* argument depends on the units being used in your simulation.

For all unit styles except *lj*, the scale argument is specified in the distance units defined by the unit style. For example, in *real* or *metal* units, if the unit cell is a unit cube with edge length 1.0, specifying scale = 3.52 would create a cubic lattice with a spacing of 3.52 Angstroms. In *cgs* units, the spacing would be 3.52 cm.

For unit style *lj*, the scale argument is the Lennard-Jones reduced density, typically written as rho*. LIGGGHTS(R)-WSU converts this value into the multiplicative factor via the formula "factor^dim = rho/rho*", where rho = N/V with V = the volume of the lattice unit cell and N = the number of basis atoms in the unit cell (described below), and dim = 2 or 3 for the dimensionality of the simulation. Effectively, this means that if LJ particles of size sigma = 1.0 are used in the simulation, the lattice of particles will be at the desired reduced density.

The *origin* option specifies how the unit cell will be shifted or translated when mapping it into the simulation box. The x,y,z values are fractional values (0.0 <= x,y,z < 1.0) meaning shift the lattice by a fraction of the lattice spacing in each dimension. The meaning of "lattice spacing" is discussed below.

The *orient* option specifies how the unit cell will be rotated when mapping it into the simulation box. The *dim* argument is one of the 3 coordinate axes in the simulation box. The other 3 arguments are the crystallographic direction in the lattice that you want to orient along that axis, specified as integers. E.g. "orient x 2 1 0" means the x-axis in the simulation box will be the [210] lattice direction. The 3 lattice directions you specify must be

mutually orthogonal and obey the right-hand rule, i.e. (X cross Y) points in the Z direction. Note that this description is really only valid for orthogonal lattices. If you are using the more general lattice style *custom* with non-orthogonal a1,a2,a3 vectors, then think of the 3 *orient* options as creating a 3x3 rotation matrix which is applied to a1,a2,a3 to rotate the original unit cell to a new orientation in the simulation box.

Several LIGGGHTS(R)-WSU commands have the option to use distance units that are inferred from "lattice spacing" in the x,y,z box directions. E.g. the region command can create a block of size 10x20x20, where 10 means 10 lattice spacings in the x direction.

The *spacing* option sets the 3 lattice spacings directly. All must be non-zero (use 1.0 for dz in a 2d simulation). The specified values are multiplied by the multiplicative factor described above that is associated with the scale factor. Thus a spacing of 1.0 means one unit cell independent of the scale factor. This option can be useful if the spacings LIGGGHTS(R)-WSU computes are inconvenient to use in subsequent commands, which can be the case for non-orthogonal or rotated lattices.

If the *spacing* option is not specified, the lattice spacings are computed by LIGGGHTS(R)-WSU in the following way. A unit cell of the lattice is mapped into the simulation box (scaled, shifted, rotated), so that it now has (perhaps) a modified size and orientation. The lattice spacing in X is defined as the difference between the min/max extent of the x coordinates of the 8 corner points of the modified unit cell. Similarly, the Y and Z lattice spacings are defined as the difference in the min/max of the y and z coordinates.

Note that if the unit cell is orthogonal with axis-aligned edges (not rotated via the *orient* keyword), then the lattice spacings in each dimension are simply the scale factor (described above) multiplied by the length of a1,a2,a3. Thus a *hex* style lattice with a scale factor of 3.0 Angstroms, would have a lattice spacing of 3.0 in x and 3*sqrt(3.0) in y.

IMPORTANT NOTE: For non-orthogonal unit cells and/or when a rotation is applied via the *orient* keyword, then the lattice spacings may be less intuitive. In particular, in these cases, there is no guarantee that the lattice spacing is an integer multiple of the periodicity of the lattice in that direction. Thus, if you create an orthogonal periodic simulation box whose size in a dimension is a multiple of the lattice spacing, and then fill it with atoms via the create_atoms command, you will NOT necessarily create a periodic system. I.e. atoms may overlap incorrectly at the faces of the simulation box.

Regardless of these issues, the values of the lattice spacings LIGGGHTS(R)-WSU calculates are printed out, so their effect in commands that use the spacings should be decipherable.

Restrictions:

The *a1,a2,a3,basis* keywords can only be used with style *custom*.

Related commands:

dimension, create_atoms, region

Default:

```
lattice none 1.0
```

For other lattice styles, the option defaults are origin = 0.0 0.0 0.0, orient = x 1 0 0, orient = y 0 1 0, orient = z 0 0 1, a1 = 1 0 0, a2 = 0 1 0, and a3 = 0 0 1.

LIGGGHTS(R)-WSU 2.X vs. LIGGGHTS(R)-WSU 1.5.3 - syntax changes

Introduction:

This is a short outline of the most important changes in LIGGGHTS(R)-WSU 2.X compared to LIGGGHTS(R)-WSU 1.5.3 regarding the syntax of major commands. The motivation for these changes was to make the script language more readable and to improve extendability (in the sense of object oriented programming) with respect to modelling approaches.

Commands covered by this tutorial:

- atom_style sphere (formerly atom_style granular)
- dump mesh/stl (formerly dump stl)
- dump mesh/vtk (formerly dump mesh/gran/VTK)
- fix heat/gran
- fix mesh/surface (formerly fix mesh/gran)
- fix mesh/surface/stress (formerly fix mesh/gran/stressanalysis)
- fix move/mesh (formerly fix move/mesh/gran)
- fix wall/gran/*
- pair_style gran/*

Changes in syntax for each command:

Changes are indicated as follows

```
OLD: old_syntax
NEW: new_syntax
```

where *old_syntax* refers to the syntax used in LIGGGHTS(R)-WSU 1.5.3 and before, and *new_syntax* refers to the syntax used in LIGGGHTS(R)-WSU 2.X

atom_style sphere (formerly atom_style granular):

```
OLD: atom_style granular
NEW: atom_style sphere
```

NOTE: For compatibility reasons, the old syntax can still be used in LIGGGHTS(R)-WSU 2.0

For details, see atom_style sphere.

dump mesh/stl:

```
OLD: dmpstl all stl 300 post/dump*.stl
NEW: dmpstl all mesh/stl 300 post/dump*.stl
```

NOTE: For compatibility reasons, the old syntax can still be used in LIGGGHTS(R)-WSU 2.0

dump mesh/vtk:

```
OLD: dmpstl all mesh/gran/VTK 300 post/dump*.stl id
NEW: dmpstl all mesh/vtk 300 post/dump*.stl id
```

NOTE: For compatibility reasons, the old syntax can still be used in LIGGGHTS(R)-WSU 2.0

---

fix heat/gran:

```
OLD: fix ID group-ID heat/gran 273.15
NEW: fix ID group-ID heat/gran initial_temperature 273.15

OLD: fix ID group-ID heat/gran 273.15 + activate area correction via fix property/global
NEW: fix ID group-ID heat/gran initial_temperature 273.15 area_correction yes
```

---

fix mesh/surface (formerly fix mesh/gran):

```
OLD: fix ID group-ID mesh/gran mesh.stl 1 1.0 0. 0. 0. 0. 0. 0.
NEW: fix ID group-ID mesh/surface file mesh.stl type 1

OLD: fix ID group-ID mesh/gran mesh.stl 1 0.001 0. 0. 0. -90. 0. 0.
NEW: fix ID group-ID mesh/surface file mesh.stl type 1 scale 0.001 rotate axis 1. 0. 0. angle -

OLD: fix ID group-ID mesh/gran mesh.stl 1 1.0 1. 2. 3. 0. 0. 0.
NEW: fix ID group-ID mesh/surface file mesh.stl type 1 move 1. 2. 3

OLD: fix ID group-ID mesh/gran mesh.stl 1 1.0 0. 0. 0. 0. 0. 0. conveyor 5. 0. 0.
NEW: fix ID group-ID mesh/surface file mesh.stl type 1 surface_vel 5. 0. 0.

OLD: fix ID group-ID mesh/gran mesh.stl 1 1.0 0. 0. 0. 0. 0. 0. rotate 0. 0. 0. 1. 0. 0. 5.
NEW: fix ID group-ID mesh/surface file mesh.stl type 1 surface_ang_vel origin 0. 0. 0. axis  1.
```

For details, see fix mesh/surface.

---

fix mesh/surface/stress (formerly fix mesh/gran/stressanalysis):

```
OLD: fix ID group-ID mesh/gran/stressanalysis mesh.stl 1 1.0 0. 0. 0. 0. 0. 0. finnie yes
NEW: fix ID group-ID fix mesh/surface/stress file mesh.stl type 1 wear finnie
```

For details, see fix mesh/surface/stress.

---

fix move/mesh (formerly fix move/mesh/gran):

```
OLD: fix ID group-ID move/mesh/gran wiggle -0.1 0. 0. 0.02 cad1 1
NEW: fix ID group-ID move/mesh mesh cad1 wiggle amplitude -0.1 0. 0. period 0.02

OLD: fix ID group-ID move/mesh/gran rotate 0. 0. 0. 0. 0. 1. 0.05 cad1 1
NEW: fix ID group-ID move/mesh mesh cad1 rotate origin 0. 0. 0. axis  0. 0. 1. period 0.05

OLD: fix ID group-ID move/mesh/gran linear 20. 20. 0. cad1 1
NEW: fix ID group-ID move/mesh mesh cad1 linear 20. 20. 0.
```

NOTES:
- The trailing "1" for the *old_syntax* was no longer used in LIGGGHTS(R)-WSU 1.5.3
- For compatibility reasons, the command name *fix move/mesh/gran* can be used in LIGGGHTS(R)-WSU 2.0. However, the syntax has to follow the LIGGGHTS(R)-WSU 2.0 syntax.

For details, see fix move/mesh.

---

fix wall/gran/*:

Changes in syntax for each command: 363

```
OLD: fix ID group-ID wall/gran/hertz/history 1 0 mesh/gran 2 cad1 cad2
NEW: fix ID group-ID wall/gran/hertz/history mesh n_meshes 2 meshes cad1 cad2


OLD: fix ID group-ID wall/gran/hertz/history 1 0 xplane -0.5 0.5 1
NEW: fix ID1 group-ID wall/gran/hertz/history type 1 xplane -0.5
     fix ID2 group-ID wall/gran/hertz/history type 1 xplane 0.5


OLD: fix ID group-ID wall/gran/hertz/history 3 0 xplane -0.5 0.5 1
NEW: fix ID1 group-ID wall/gran/hertz/history primitive type 1 xplane -0.5 rolling_friction cdt
     fix ID2 group-ID wall/gran/hertz/history primitive type 1 xplane  0.5 rolling_friction cdt


OLD: fix ID group-ID wall/gran/hertz/history 1 1 xplane -0.5 0.5 1
NEW: fix ID group-ID wall/gran/hertz/history primitive type 1 xplane -0.5 0.5 cohesion sjkr


OLD: fix ID group-ID wall/gran/hertz/history 0 0 zcylinder 0.05 1
NEW: fix ID group-ID wall/gran/hertz/history primitive type 1 zcylinder 0.05 0. 0. tangential_d
```

NOTES:
- Same applies for hooke/history, hooke, hooke/history/simple, hertz/history/simple
- *sjkr* stands for 'simplified JKR (Johnson-Kendall-Roberts)' model, and *cdt* for 'constant directional torque' model
- styles *xplane*, *yplane*, *zplane* take only one arg now (the wall position), if you want two walls you have to use two fix commands
- in addition to the existing style *zcylinder*, there is now *xcylinder* and *ycylinder* as well
- *xcylinder ycylinder* and *zcylinder* take 3 args: the cylinder radius, and the location of the axis in the other two dimensions

For details, see fix wall/gran.

---

pair_style gran/*:

```
OLD: pair_style gran/hertz/history 1 0
NEW: pair_style gran/hertz/history


OLD: pair_style gran/hertz/history 3 0
NEW: pair_style gran/hertz/history rolling_friction cdt


OLD: pair_style gran/hertz/history 1 1
NEW: pair_style gran/hertz/history cohesion sjkr


OLD: pair_style gran/hertz/history 0 0
NEW: pair_style gran/hertz/history tangential_damping off
```

NOTES:
- Same applies for hooke/history, hooke, hooke/history/simple, hertz/history/simple
- *sjkr* stands for "simplified JKR (Johnson-Kendall-Roberts)" model, and *cdt* for "constant directional torque" model

For details, see pair_style gran.

Changes in syntax for each command: 364

LIGGGHTS(R)-WSU 3.X vs. LIGGGHTS(R)-WSU 2.X - syntax changes

Introduction:

This is a short outline of the most important changes in LIGGGHTS(R)-WSU 3.X compared to LIGGGHTS(R)-WSU 2.X regarding the syntax of major commands. The motivation for these changes was to make the script language more readable and to improve extendability (in the sense of object oriented programming) with respect to modelling approaches.

Commands covered by this tutorial:

- fix wall/gran/*
- pair_style gran/*

Changes in syntax for each command:

Changes are indicated as follows

```
OLD: old_syntax
NEW: new_syntax
```

where *old_syntax* refers to the syntax used in LIGGGHTS(R)-WSU 2.X and before, and *new_syntax* refers to the syntax used in LIGGGHTS(R)-WSU 3.X.

---

fix wall/gran/*:

```
OLD: fix ID group-ID wall/gran/* WALL-OPTIONS
NEW: fix ID group-ID wall/gran MODEL-SELECTION WALL-OPTIONS MODEL-SETTINGS
```

MODEL-SELECTION

MODEL-SELECTION = model *M* [tangential *T*] [cohesion *C*] [rolling_friction *R*]

M = hooke | hooke/stiffness | hooke/hysteresis | hertz | hertz/stiffness

T = **no_history** | history

C = **off** | sjkr | sjkr2 | hamaker

R = **off** | cdt | epsd

MODEL-SETTINGS

MODEL-SELECTION = [tangential_damping (**on**|off)] [absolute_damping (on|**off**)] [viscous (on|**off**)]

Examples

```
OLD: fix ID group-ID wall/gran/hertz/history mesh n_meshes 2 meshes cad1 cad2
NEW: fix ID group-ID wall/gran model hertz tangential history mesh n_meshes 2 meshes cad1 cad2


OLD: fix ID1 group-ID wall/gran/hooke/history type 1 xplane -0.5
NEW: fix ID1 group-ID wall/gran model hooke tangential history type 1 xplane -0.5


OLD: fix ID1 group-ID wall/gran/hertz/history primitive type 1 xplane -0.5 rolling_friction cdt
NEW: fix ID1 group-ID wall/gran model hertz tangential history rolling_friction cdt primitive t


OLD: fix ID group-ID wall/gran/hertz/history primitive type 1 xplane -0.5 0.5 cohesion sjkr
NEW: fix ID group-ID wall/gran model hertz tangential history cohesion sjkr primitive type 1 xp


OLD: fix ID group-ID wall/gran/hertz/history primitive type 1 zcylinder 0.05 0. 0. tangential_d
NEW: fix ID group-ID wall/gran model hertz tangential history primitive type 1 zcylinder 0.05 0
```

pair_style gran/*:

```
OLD: pair_style gran/* MODEL-SETTINGS
NEW: pair_style gran   MODEL-SELECTION MODEL-SETTINGS
```

MODEL-SELECTION

Same as in fix wall/gran/*

MODEL-SETTINGS

Same as in fix wall/gran/*

Examples

```
OLD: pair_style gran/hertz/history
NEW: pair_style gran model hertz tangential history


OLD: pair_style gran/hertz/history rolling_friction cdt
NEW: pair_style gran model hertz tangential history rolling_friction cdt


OLD: pair_style gran/hertz/history cohesion sjkr
NEW: pair_style gran model hertz tangential history cohesion sjkr


OLD: pair_style gran/hertz/history tangential_damping off
OLD: pair_style gran model hertz tangential history tangential_damping off
```

log command

Syntax:

```
log file keyword
```

- file = name of new logfile
- keyword = *append* if output should be appended to logfile (optional)

Examples:

```
log log.equil
log log.equil append
```

Description:

This command closes the current LIGGGHTS(R)-WSU 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 the optional keyword *append* is specified, then output will be appended to an existing log file, instead of overwriting it.

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 LIGGGHTS(R)-WSU run. The name of the initial log file can also be set by the command-line switch -log. See Section_start 6 for details.

Restrictions: none

Related commands: none

Default:

The default LIGGGHTS(R)-WSU log file is named log.lammps

LIGGGHTS(R)-WSU Documentation, Version 3.X

LIGGGHTS(R)-WSU DEM simulation engine

released by DCS Computing Gmbh, Linz, Austria, www.dcs-computing.com , office@dcs-computing.com

LIGGGHTS(R)-WSU is open-source, distributed under the terms of the GNU Public License, version 2 or later. LIGGGHTS(R)-WSU is part of CFDEM(R)project: www.ligghts.com | www.cfdem.com

Core developer and main author: Christoph Kloss, christoph.kloss@dcs-computing.com

LIGGGHTS(R)-WSU is an Open Source Discrete Element Method Particle Simulation Software, distributed by DCS Computing GmbH, Linz, Austria. LIGGGHTS (R) and CFDEM(R) are registered trade marks of DCS Computing GmbH, the producer of the LIGGGHTS (R) software and the CFDEM(R)coupling software See http://www.cfdem.com/terms-trademark-policy for details.

LIGGGHTS (R) Version info:

All LIGGGHTS (R) versions are based on a specific version of LIGGGHTS (R), as printed in the file src/version.h LIGGGHTS (R) versions are identidied by a version number (e.g. '3.0'), a branch name (which is 'LIGGGHTS(R)-WSU' for your release of LIGGGHTS), compilation info (date / time stamp and user name), and a LAMMPS version number (which is the LAMMPS version that the LIGGGHTS(R)-WSU release is based on). The LAMMPS "version" is the date when it was released, such as 1 May 2010.

If you browse the HTML doc pages on the LIGGGHTS(R)-WSU WWW site, they always describe the most current version of LIGGGHTS(R)-WSU. If you browse the HTML doc pages included in your tarball, they describe the version you have.

LIGGGHTS (R) and its ancestor LAMMPS:

LAMMPS is a classical molecular dynamics simulation code designed to run efficiently on parallel computers. It was developed at Sandia National Laboratories, a US Department of Energy facility, with funding from the DOE. It is an open-source code, distributed freely under the terms of the GNU Public License (GPL). The primary developers of LAMMPS are Steve Plimpton, Aidan Thompson, and Paul Crozier. The LAMMPS WWW Site at http://lammps.sandia.gov has more information about LAMMPS.

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

Once you are familiar with LIGGGHTS(R)-WSU, you may want to bookmark this page since it gives quick access to documentation for all LIGGGHTS(R)-WSU commands.

LIGGGHTS (R) and its ancestor LAMMPS:

mass command

Syntax:

```
mass I value
```

- I = atom type (see asterisk form below)
- value = mass

Examples:

```
mass 1 1.0
mass * 62.5
mass 2* 62.5
```

Description:

Set the mass for all atoms of one or more atom types. Per-type mass values can also be set in the read_data data file using the "Masses" keyword. See the units command for what mass units to use.

The I index can be specified in one of two ways. An explicit numeric value can be used, as in the 1st example above. Or a wild-card asterisk can be used to set the mass for multiple atom types. This takes the form "*" or "*n" or "n*" or "m*n". If N = the number of atom types, then an asterisk with no numeric values means all types from 1 to N. A leading asterisk means all types from 1 to n (inclusive). A trailing asterisk means all types from n to N (inclusive). A middle asterisk means all types from m to n (inclusive).

A line in a data file that follows the "Masses" keyword specifies mass using the same format as the arguments of the mass command in an input script, except that no wild-card asterisk can be used. For example, under the "Masses" section of a data file, the line that corresponds to the 1st example above would be listed as

```
1 1.0
```

Note that the mass command can only be used if the atom style requires per-type atom mass to be set. Currently, all but the *sphere* and *ellipsoid* and *peri* styles do. They require mass to be set for individual particles, not types. Per-atom masses are defined in the data file read by the read_data command, or set to default values by the create_atoms command. Per-atom masses can also be set to new values by the set mass or set density commands.

Also note that pair_style eam defines the masses of atom types in the EAM potential file, in which case the mass command is normally not used.

If you define a hybrid atom style which includes one (or more) sub-styles which require per-type mass and one (or more) sub-styles which require per-atom mass, then you must define both. However, in this case the per-type mass will be ignored; only the per-atom mass will be used by LIGGGHTS(R)-WSU.

Restrictions:

This command must come after the simulation box is defined by a read_data, read_restart, or create_box command.

All masses must be defined before a simulation is run. They must also all be defined before a velocity or fix shake command is used.

The mass assigned to any type or atom must be > 0.0.

Related commands: none

Default: none

neighbor command

Syntax:

```
neighbor skin style
```

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

Examples:

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

Description:

This command sets parameters that affect the building of pairwise neighbor lists. All atom pairs within a neighbor cutoff distance equal to the their force cutoff plus the *skin* distance are stored in the list. Typically, the larger the skin distance, the less often neighbor lists need to be built, but more pairs must be checked for possible force interactions every timestep. The default value for *skin* depends on the choice of units for the simulation; see the default values below.

The *skin* distance is also used to determine how often atoms migrate to new processors if the *check* option of the [neigh_modify](#) command is set to *yes*. Atoms are migrated (communicated) to new processors on the same timestep that neighbor lists are re-built.

The *style* value selects what algorithm is used to build the list. The *bin* style creates the list by binning which is an operation that scales linearly with N/P, the number of atoms per processor where N = total number of atoms and P = number of processors. It is almost always faster than the *nsq* style which scales as (N/P)^2. For unsolvated small molecules in a non-periodic box, the *nsq* choice can sometimes be faster. Either style should give the same answers.

The *multi* style is a modified binning algorithm that is useful for systems with a wide range of cutoff distances, e.g. due to different size particles. For the *bin* style, the bin size is set to 1/2 of the largest cutoff distance between any pair of atom types and a single set of bins is defined to search over for all atom types. This can be inefficient if one pair of types has a very long cutoff, but other type pairs have a much shorter cutoff. For style *multi* the bin size is set to 1/2 of the shortest cutoff distance and multiple sets of bins are defined to search over for different atom types. This imposes some extra setup overhead, but the searches themselves may be much faster for the short-cutoff cases. See the [communicate multi](#) command for a communication option option that may also be beneficial for simulations of this kind.

The [neigh_modify](#) command has additional options that control how often neighbor lists are built and which pairs are stored in the list.

When a run is finished, counts of the number of neighbors stored in the pairwise list and the number of times neighbor lists were built are printed to the screen and log file. See [this section](#) for details.

Restrictions: none

Related commands:

[neigh_modify](#), [units](#), [communicate](#)

Default:

0.3 bin for units = lj, skin = 0.3 sigma
2.0 bin for units = real or metal, skin = 2.0 Angstroms
0.001 bin for units = si, skin = 0.001 meters = 1.0 mm
0.1 bin for units = cgs, skin = 0.1 cm = 1.0 mm

neigh_modify command

Syntax:

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

- one or more keyword/value pairs may be listed

```
keyword = delay or every or check or once or cluster or include or exclude or page or one
  delay value = N
    N = delay building until this many steps since last build
  every value = M
    M = build neighbor list every this many steps
  check value = yes or no
    yes = only build if some atom has moved half the skin distance or more
    no = always build on 1st step that every and delay are satisfied
  once
    yes = only build neighbor list once at start of run and never rebuild
    no = rebuild neighbor list according to other settings
  cluster
    yes = check bond,angle,etc neighbor list for nearby clusters
    no = do not check bond,angle,etc neighbor list for nearby clusters
  include value = group-ID
    group-ID = only build pair neighbor lists for atoms in this group
  exclude values:
    type M N
      M,N = exclude if one atom in pair is type M, other is type N
    group group1-ID group2-ID
      group1-ID,group2-ID = exclude if one atom is in 1st group, other in 2nd
    molecule group-ID
      groupname = exclude if both atoms are in the same molecule and in the same group
    none
      delete all exclude settings
  page value = N
    N = number of pairs stored in a single neighbor page
  one value = N
    N = max number of neighbors of one atom
  contact_distance_factor value = N
    N = contact distance factor used to extend the range of granular neighbor lists (must
  binsize value = size
    size = bin size for neighbor list construction (distance units)
```

Examples:

```
neigh_modify every 2 delay 10 check yes page 100000
neigh_modify exclude type 2 3
neigh_modify exclude group frozen frozen check no
neigh_modify exclude group residue1 chain3
neigh_modify exclude molecule rigid
neigh_modify delay 0 contact_distance_factor 1.5
```

Description:

This command sets parameters that affect the building and use of pairwise neighbor lists.

The *every*, *delay*, *check*, and *once* options affect how often lists are built as a simulation runs. The *delay* setting means never build a new list until at least N steps after the previous build. The *every* setting means build the list every M steps (after the delay has passed). If the *check* setting is *no*, the list is built on the 1st

step that satisfies the *delay* and *every* settings. If the *check* setting is *yes*, then the list is only built on a particular step if some atom has moved more than half the skin distance (specified in the neighbor command) since the last build. If the *once* setting is yes, then the neighbor list is only built once at the beginning of each run, and never rebuilt. This should only be done if you are certain atoms will not move far enough that the list should be rebuilt. E.g. running a simulation of a cold crystal. Note that it is not that expensive to check if neighbor lists should be rebuilt.

When the rRESPA integrator is used (see the run_style command), the *every* and *delay* parameters refer to the longest (outermost) timestep.

The *contact_distance_factor* setting can be used to increase the range of granular neighbor lists. When *contact_distance_factor* > 1.0, instead of the standard criterion ri+rj+skin < distance, LIGGGHTS(R)-WSU is checking for *contact_distance_factor* *(ri+rj)+skin < distance to decided if a pair of granular particles goes into a neighbor list.

The *cluster* option does a sanity test every time neighbor lists are built for bond, angle, dihedral, and improper interactions, to check that each set of 2, 3, or 4 atoms is a cluster of nearby atoms. It does this by computing the distance between pairs of atoms in the interaction and insuring they are not further apart than half the periodic box length. If they are, an error is generated, since the interaction would be computed between far-away atoms instead of their nearby periodic images. The only way this should happen is if the pairwise cutoff is so short that atoms that are part of the same interaction are not communicated as ghost atoms. This is an unusual model (e.g. no pair interactions at all) and the problem can be fixed by use of the [communicate cutoff](communicate) command. Note that to save time, the default *cluster* setting is *no*, so that this check is not performed.

The *include* option limits the building of pairwise neighbor lists to atoms in the specified group. This can be useful for models where a large portion of the simulation is particles that do not interact with other particles or with each other via pairwise interactions. The group specified with this option must also be specified via the [atom_modify first](atom_modify) command.

The *exclude* option turns off pairwise interactions between certain pairs of atoms, by not including them in the neighbor list. These are sample scenarios where this is useful:

- In crack simulations, pairwise interactions can be shut off between 2 slabs of atoms to effectively create a crack.
- When a large collection of atoms is treated as frozen, interactions between those atoms can be turned off to save needless computation. E.g. Using the [fix setforce](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](fix_rigid) command for more details.

The *exclude type* option turns off the pairwise interaction if one atom is of type M and the other of type N. M can equal N. The *exclude group* option turns off the interaction if one atom is in the first group and the other is the second. Group1-ID can equal group2-ID. The *exclude molecule* option turns off the interaction if both atoms are in the specified group and in the same molecule, as determined by their molecule ID.

Each of the exclude options can be specified multiple times. The *exclude type* option is the most efficient option to use; it requires only a single check, no matter how many times it has been specified. The other exclude options are more expensive if specified multiple times; they require one check for each time they have been specified.

Note that the exclude options only affect pairwise interactions; see the delete_bonds command for information on turning off bond interactions.

The *page* and *one* options affect how memory is allocated for the neighbor lists. For most simulations the

neigh_modify command   376

default settings for these options are fine, but if a very large problem is being run or a very long cutoff is being used, these parameters can be tuned. The indices of neighboring atoms are stored in "pages", which are allocated one after another as they fill up. The size of each page is set by the *page* value. A new page is allocated when the next atom's neighbors could potentially overflow the list. This threshold is set by the *one* value which tells LIGGGHTS(R)-WSU the maximum number of neighbor's one atom can have.

IMPORTANT NOTE: LIGGGHTS(R)-WSU can crash without an error message if the number of neighbors for a single particle is larger than the *page* setting, which means it is much, much larger than the *one* setting. This is because LIGGGHTS(R)-WSU doesn't error check these limits for every pairwise interaction (too costly), but only after all the particle's neighbors have been found. This problem usually means something is very wrong with the way you've setup your problem (particle spacing, cutoff length, neighbor skin distance, etc). If you really expect that many neighbors per particle, then boost the *one* and *page* settings accordingly.

The *binsize* option allows you to specify what size of bins will be used in neighbor list construction to sort and find neighboring atoms. By default, for [neighbor style bin](#), LIGGGHTS(R)-WSU uses bins that are 1/2 the size of the maximum pair cutoff. For [neighbor style multi](#), the bins are 1/2 the size of the minimum pair cutoff. Typically these are good values values for minimizing the time for neighbor list construction. This setting overrides the default. If you make it too big, there is little overhead due to looping over bins, but more atoms are checked. If you make it too small, the optimal number of atoms is checked, but bin overhead goes up. If you set the binsize to 0.0, LIGGGHTS(R)-WSU will use the default binsize of 1/2 the cutoff.

Restrictions:

If the "delay" setting is non-zero, then it must be a multiple of the "every" setting.

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

The value of the *page* setting must be at least 10x larger than the *one* setting. This insures neighbor pages are not mostly empty space.

Related commands:

[neighbor](#), [delete_bonds](#)

Default:

The option defaults are delay = 10, every = 1, check = yes, once = no, cluster = no, include = all, exclude = none, page = 100000, one = 2000, and binsize = 0.0.

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.

LIGGGHTS(R)-WSU 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.

Some commands may not support newton "on". You will see an error message in this case when running the simulation.

Related commands:

run_style respa

Default:

```
newton on
```

next command

Syntax:

```
next variables
```

- variables = one or more variable names

Examples:

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

Description:

This command is used with variables defined by the [variable](#) command. It assigns the next value to the variable from the list of values defined for that variable by the [variable](#) command. Thus when that variable is subsequently substituted for in an input script command, the new value is used.

See the [variable](#) command for info on how to define and use different kinds of variables in LIGGGHTS(R)-WSU input scripts. If a variable name is a single lower-case character from "a" to "z", it can be used in an input script command as $a or $z. If it is multiple letters, it can be used as ${myTemp}.

If multiple variables are used as arguments to the *next* command, then all must be of the same variable style: *index*, *loop*, *file*, *universe*, or *uloop*. An exception is that *universe-* and *uloop*-style variables can be mixed in the same *next* command.

All the variables specified with the next command are incremented by one value from their respective list of values. A *file*-style variable reads the next line from its associated file. An *atomfile*-style variable reads the next set of lines (one per atom) from its associated file. *String-* or *atom-* or *equal-* or *world*-style variables cannot be used with the the next command, since they only store a single value.

When any of the variables in the next command has no more values, a flag is set that causes the input script to skip the next [jump](#) command encountered. This enables a loop containing a next command to exit. As explained in the [variable](#) command, the variable that has exhausted its values is also deleted. This allows it to be used and re-defined later in the input script. *File*-style and *atomfile*-style variables are exhausted when the end-of-file is reached.

When the next command is used with *index-* or *loop*-style variables, the next value is assigned to the variable for all processors. When the next command is used with *file*-style variables, the next line is read from its file and the string assigned to the variable. When the next command is used with *atomfile*-style variables, the next set of per-atom values is read from its file and assigned to the variable. When the next command is used with *universe-* or *uloop*-style variables, the next value is assigned to whichever processor partition executes the command first. All processors in the partition are assigned the same value. Running LIGGGHTS(R)-WSU on multiple partitions of processors via the "-partition" command-line switch is described in [this section](#) of the manual. *Universe-* and *uloop*-style variables are incremented using the files "tmp.lammps.variable" and "tmp.lammps.variable.lock" which you will see in your directory during such a LIGGGHTS(R)-WSU run.

Here is an example of running a series of simulations using the next command with an *index*-style variable. If this input script is named in.polymer, 8 simulations would be run using data files from directories run1 thru run8.

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

If the variable "d" were of style *universe*, and the same in.polymer input script were run on 3 partitions of processors, then the first 3 simulations would begin, one on each set of processors. Whichever partition finished first, it would assign variable "d" the 4th value and run another simulation, and so forth until all 8 simulations were finished.

Jump and next commands can also be nested to enable multi-level loops. For example, this script will run 15 simulations in a double loop.

```
variable i loop 3
  variable j loop 5
  clear
  ...
  read_data data.polymer.$i$j
  print Running simulation $i.$j
  run 10000
  next j
  jump in.script
next i
jump in.script
```

Here is an example of a double loop which uses the [if] and [jump] commands to break out of the inner loop when a condition is met, then continues iterating thru the outer loop.

```
label       loopa
variable    a loop 5
  label     loopb
  variable  b loop 5
  print     "A,B = $a,$b"
  run       10000
  if        $b > 2 then "jump in.script break"
  next      b
  jump      in.script loopb
label       break
variable    b delete

next        a
jump        in.script loopa
```

Restrictions: none

Related commands:

[jump], [include], [shell], [variable],

Default: none

orient command

Syntax:

```
orient dim i j k
```

- dim = *x* or *y* or *z*
- i,j,k = orientation of lattice that is along box direction dim

Examples:

```
orient x 1 1 0
orient y -1 1 0
orient z 0 0 1
```

Description:

Specify the orientation of a cubic lattice along simulation box directions *x* or *y* or *z*. These 3 basis vectors are used when the create_atoms command generates a lattice of atoms.

The 3 basis vectors B1, B2, B3 must be mutually orthogonal and form a right-handed system such that B1 cross B2 is in the direction of B3.

The basis vectors should be specified in an irreducible form (smallest possible integers), though LIGGGHTS(R)-WSU 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 asterisk form below)
- args = coefficients for one or more pairs of atom types

Examples:

```
pair_coeff 1 2 1.0 1.0 2.5
pair_coeff 2 * 1.0 1.0
pair_coeff 3* 1*2 1.0 1.0 2.5
pair_coeff * * 1.0 1.0
pair_coeff * * nialhjea 1 1 2
pair_coeff * 3 morse.table ENTRY1
pair_coeff 1 2 lj/cut 1.0 1.0 2.5 (for pair_style hybrid)
```

Description:

Specify the pairwise force field coefficients for one or more pairs of atom types. The number and meaning of the coefficients depends on the pair style. Pair coefficients can also be set in the data file read by the [read_data](#) command or in a restart file.

I and J can be specified in one of two ways. Explicit numeric values can be used for each, as in the 1st example above. I <= J is required. LIGGGHTS(R)-WSU sets the coefficients for the symmetric J,I interaction to the same values.

A wildcard asterisk can be used in place of or in conjunction with the I,J arguments to set the coefficients for multiple pairs of atom types. This takes the form "*" or "*n" or "n*" or "m*n". If N = the number of atom types, then an asterisk with no numeric values means all types from 1 to N. A leading asterisk means all types from 1 to n (inclusive). A trailing asterisk means all types from n to N (inclusive). A middle asterisk means all types from m to n (inclusive). Note that only type pairs with I <= J are considered; if asterisks imply type pairs where J < I, they are ignored.

Note that a pair_coeff command can override a previous setting for the same I,J pair. For example, these commands set the coeffs for all I,J pairs, then overwrite the coeffs for just the I,J = 2,3 pair:

```
pair_coeff * * 1.0 1.0 2.5
pair_coeff 2 3 2.0 1.0 1.12
```

A line in a data file that specifies pair coefficients uses the exact same format as the arguments of the pair_coeff command in an input script, with the exception of the I,J type arguments. In each line of the "Pair Coeffs" section of a data file, only a single type I is specified, which sets the coefficients for type I interacting with type I. This is because the section has exactly N lines, where N = the number of atom types. For this reason, the wild-card asterisk should also not be used as part of the I argument. Thus in a data file, the line corresponding to the 1st example above would be listed as

```
2 1.0 1.0 2.5
```

For many potentials, if coefficients for type pairs with I != J are not set explicitly by a pair_coeff command, the values are inferred from the I,I and J,J settings by mixing rules; see the [pair_modify](#) command for a

discussion. Details on this option as it pertains to individual potentials are described on the doc page for the potential.

Many pair styles, typically for many-body potentials, use tabulated potential files as input, when specifying the pair_coeff command. Potential files provided with LIGGGHTS(R)-WSU are in the potentials directory of the distribution. For some potentials, such as EAM, other archives of suitable files can be found on the Web. They can be used with LIGGGHTS(R)-WSU so long as they are in the format LIGGGHTS(R)-WSU expects, as discussed on the individual doc pages.

When a pair_coeff command using a potential file is specified, LIGGGHTS(R)-WSU looks for the potential file in 2 places. First it looks in the location specified. E.g. if the file is specified as "niu3.eam", it is looked for in the current working directory. If it is specified as "../potentials/niu3.eam", then it is looked for in the potentials directory, assuming it is a sister directory of the current working directory. If the file is not found, it is then looked for in the directory specified by the LAMMPS_POTENTIALS environment variable. Thus if this is set to the potentials directory in the LIGGGHTS(R)-WSU distro, then you can use those files from anywhere on your system, without copying them into your working directory. Environment variables are set in different ways for different shells. Here are example settings for

```
csh, tcsh:
% setenv LAMMPS_POTENTIALS /path/to/lammps/potentials

bash:
% export LAMMPS_POTENTIALS=/path/to/lammps/potentials

Windows:
% set LAMMPS_POTENTIALS="C:\Path to LAMMPS\Potentials
```

The full list of pair styles defined in LIGGGHTS(R)-WSU is on this page.

Restrictions:

This command must come after the simulation box is defined by a read_data, read_restart, or create_box command.

Related commands:

pair_style, pair_modify, read_data, read_restart, pair_write

Default: none

pair_style gran command

Syntax:

```
pair_style model_type model_name model_keyword model_value
```

- zero or more model_type/model_name pairs may be appended. They must be appended in the following order (!)

```
model values = described here
  tangential values = described here
  cohesion values = described here
  rolling_friction values = described here
  surface values = described here
```

- following the model_type/model_name pairs, zero or more model_keyword/model_valuezero pairs may be appended in arbitrary order

```
model_type/model_name pairs = described for each model separately here
```

Examples:

```
pair_style gran model hooke tangential history
pair_style gran model hertz tangential history rolling_friction cdt
pair_style gran model hertz tangential no_history cohesion sjkr
```

General description:

The *gran* styles use the following formula for the frictional force between two granular particles, when the distance r between two particles of radii Ri and Rj is less than their contact distance d = Ri + Rj. Typically, there is no force between the particles when r > d.

$$F = \left(k_n \; \underbrace{\delta n_{ij}}_{\substack{normal \\ overlap}} \; - \gamma_n \; \underbrace{vn_{ij}}_{\substack{normal \\ relative\ vel.}} \right) + \left(k_t \; \underbrace{\delta t_{ij}}_{\substack{tangential \\ overlap}} \; - \gamma_t \; \underbrace{vt_{ij}}_{\substack{tangential \\ relative\ v}} \right.$$
$$\underbrace{\qquad\qquad\qquad\qquad\qquad}_{normal\ force} \qquad \underbrace{\qquad\qquad\qquad\qquad}_{tangential\ force}$$

The tangential overlap is truncated to fulfil $F_t \leq X_\mu F_n$

The quantities in the equations are as follows:

- delta_n = d - r = overlap distance of 2 particles
- k_n = elastic constant for normal contact
- k_t = elastic constant for tangential contact

- gamma_n = viscoelastic damping constant for normal contact
- gamma_t = viscoelastic damping constant for tangential contact
- delta_t = tangential displacement vector between 2 spherical particles

In the first term is the normal force between the two particles and the second term is the tangential force. The normal force has 2 terms, a contact force and a damping force. The tangential force also has 2 terms: a shear force and a damping force. The shear force is a "history" effect that accounts for the tangential displacement ("tangential overlap") between the particles for the duration of the time they are in contact.

The concrete implementation for k_n, k_t, gamma_n, gamma_t and the shear history depend on the concrete models as chosen by the user. They are described on separate doc pages here

Also, other models may add additional forces or torques on the particles, such as cohesive or rolling friction forces. These are also described on separate doc pages here

IMPORTANT NOTE: The order of model keywords is important, you have to stick to the order as outlined in the "Syntax" section of this doc page.

General comments:

For granular styles there are no additional coefficients to set for each pair of atom types via the pair_coeff command. All settings are global and are made via the pair_style command. However you must still use the pair_coeff for all pairs of granular atom types. For example the command

```
pair_coeff * *
```

should be used if all atoms in the simulation interact via a granular potential (i.e. one of the pair styles above is used). If a granular potential is used as a sub-style of pair_style hybrid, then specific atom types can be used in the pair_coeff command to determine which atoms interact via a granular potential.

Mixing, shift, table, tail correction, restart, rRESPA info:

The pair_modify mix, shift, table, and tail options are not relevant for granular pair styles.

These pair styles write their information to binary restart files, so a pair_style command does not need to be specified in an input script that reads a restart file.

IMPORTANT NOTE: The material properties are not written to restart files! Thus, if you restart a simulation, you have to re-define them (by using the fixes mentioned above).

These pair styles can only be used via the pair keyword of the run_style respa command. They do not support the inner, middle, outer keywords.

Restrictions:

These pair styles require that atoms store torque and angular velocity (omega) as defined by the atom_style. They also require a per-particle radius is stored. The *sphere* or *granular* atom style does all of this.

This pair style requires you to use the communicate vel yes option so that velocites are stored by ghost atoms.

Only unit system that are self-consistent (si, cgs, lj) can be used with this pair style.

Related commands:

pair_coeff Models for use with this command are described here

Default:

model = 'hertz' *tangential* = 'off' *rolling_friction* = 'off' *cohesion* = 'off' *surface* = 'default'

pair_style hybrid command

pair_style hybrid/overlay command

Syntax:

```
pair_style hybrid style1 args style2 args ...
pair_style hybrid/overlay style1 args style2 args ...
```

- style1,style2 = list of one or more pair styles and their arguments

Examples:

```
none
```

Description:

The *hybrid* and *hybrid/overlay* styles enable the use of multiple pair styles in one simulation. With the *hybrid* style, exactly one pair style is assigned to each pair of atom types. With the *hybrid/overlay* style, one or more pair styles can be assigned to each pair of atom types. The assignment of pair styles to type pairs is made via the [pair_coeff](#) command.

All pair styles that will be used are listed as "sub-styles" following the *hybrid* or *hybrid/overlay* keyword, in any order. Each sub-style's name is followed by its usual arguments, as illustrated in the example above. See the doc pages of individual pair styles for a listing and explanation of the appropriate arguments.

Note that an individual pair style can be used multiple times as a sub-style. For efficiency this should only be done if your model requires it.

In the pair_coeff commands, the name of a pair style must be added after the I,J type specification, with the remaining coefficients being those appropriate to that style. If the pair style is used multiple times in the pair_style command with, then an additional numeric argument must also be included which is the number from 1 to M where M is the number of times the sub-style was listed in the pair style command. The extra number indicates which instance of the sub-style these coefficients apply to.

If pair coefficients are specified in the data file read via the [read_data](#) command, then the same rule applies.

For the *hybrid* style, each atom type pair I,J is assigned to exactly one sub-style. Just as with a simulation using a single pair style, if you specify the same atom type pair in a second pair_coeff command, the previous assignment will be overwritten.

For the *hybrid/overlay* style, each atom type pair I,J can be assigned to one or more sub-styles. If you specify the same atom type pair in a second pair_coeff command with a new sub-style, then the second sub-style is added to the list of potentials that will be calculated for two interacting atoms of those types. If you specify the same atom type pair in a second pair_coeff command with a sub-style that has already been defined for that pair of atoms, then the new pair coefficients simply override the previous ones, as in the normal usage of the pair_coeff command.

Coefficients must be defined for each pair of atoms types via the [pair_coeff](#) command as described above, or in the data file or restart files read by the [read_data](#) or [read_restart](#) commands, or by mixing as described below.

For both the *hybrid* and *hybrid/overlay* styles, every atom type pair I,J (where I <= J) must be assigned to at least one sub-style via the pair_coeff command as in the examples above, or in the data file read by the read_data, or by mixing as described below.

If you want there to be no interactions between a particular pair of atom types, you have 3 choices. You can assign the type pair to some sub-style and use the neigh_modify exclude type command. You can assign it to some sub-style and set the coefficients so that there is effectively no interaction. Or, for *hybrid* and *hybrid/overlay* simulations, you can use this form of the pair_coeff command in your input script:

```
pair_coeff     2 3 none
```

or this form in the "Pair Coeffs" section of the data file:

```
3 none
```

If an assignment to *none* is made in a simulation with the *hybrid/overlay* pair style, it wipes out all previous assignments of that atom type pair to sub-styles.

Note that you may need to use an atom_style hybrid command in your input script, if atoms in the simulation will need attributes from several atom styles, due to using multiple pair potentials.

You can use the pair_coeff none setting or the neigh_modify exclude command to exclude certain type pairs from the neighbor list

---

Since the *hybrid* and *hybrid/overlay* styles delegate computation to the individual sub-styles, the suffix versions of the *hybrid* and *hybrid/overlay* styles are used to propagate the corresponding suffix to all sub-styles, if those versions exist. Otherwise the non-accelerated version will be used.

Mixing, shift, table, tail correction, restart, rRESPA info:

Any pair potential settings made via the pair_modify command are passed along to all sub-styles of the hybrid potential.

For atom type pairs I,J and I != J, if the sub-style assigned to I,I and J,J is the same, and if the sub-style allows for mixing, then the coefficients for I,J can be mixed. This means you do not have to specify a pair_coeff command for I,J since the I,J type pair will be assigned automatically to the I,I sub-style and its coefficients generated by the mixing rule used by that sub-style. For the *hybrid/overlay* style, there is an additional requirement that both the I,I and J,J pairs are assigned to a single sub-style. See the "pair_modify" command for details of mixing rules. See the See the doc page for the sub-style to see if allows for mixing.

The hybrid pair styles supports the pair_modify shift, table, and tail options for an I,J pair interaction, if the associated sub-style supports it.

For the hybrid pair styles, the list of sub-styles and their respective settings are written to binary restart files, so a pair_style command does not need to specified in an input script that reads a restart file. However, the coefficient information is not stored in the restart file. Thus, pair_coeff commands need to be re-specified in the restart input script.

These pair styles support the use of the *inner*, *middle*, and *outer* keywords of the run_style respa command, if their sub-styles do.

Restrictions: none

Related commands:

pair_style hybrid/overlay command

pair_coeff

Default: none

pair_coeff

Default: none

pair_style none command

Syntax:

```
pair_style none
```

Examples:

```
pair_style none
```

Description:

Using a pair style of none means pair forces are not computed.

With this choice, the force cutoff is 0.0, which means that only atoms within the neighbor skin distance (see the [neighbor](#) command) are communicated between processors. You must insure the skin distance is large enough to acquire atoms needed for computing bonds, angles, etc.

A pair style of *none* will also prevent pairwise neighbor lists from being built. However if the [neighbor](#) style is *bin*, data structures for binning are still allocated. If the neighbor skin distance is small, then these data structures can consume a large amount of memory. So you should either set the neighbor style to *nsq* or set the skin distance to a larger value.

Restrictions: none

Related commands: none

Default: none

pair_style soft command

Syntax:

```
pair_style soft cutoff
```

- cutoff = global cutoff for soft interactions (distance units)

Examples:

```
pair_style soft 2.5
pair_coeff * * 10.0
pair_coeff 1 1 10.0 3.0

pair_style soft 2.5
pair_coeff * * 0.0
variable prefactor equal ramp(0,30)
fix 1 all adapt 1 pair soft a * * v_prefactor
```

Description:

Style *soft* computes pairwise interactions with the formula

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

It is useful for pushing apart overlapping atoms, since it does not blow up as r goes to 0. A is a pre-factor that can be made to vary in time from the start to the end of the run (see discussion below), e.g. to start with a very soft potential and slowly harden the interactions over time. Rc is the cutoff. See the fix nve/limit command for another way to push apart overlapping atoms.

The following coefficients must be defined for each pair of atom types via the pair_coeff command as in the examples above, or in the data file or restart files read by the read_data or read_restart commands, or by mixing as described below:

- A (energy units)
- cutoff (distance units)

The last coefficient is optional. If not specified, the global soft cutoff is used.

IMPORTANT NOTE: The syntax for pair_coeff with a single A coeff is different in the current version of LIGGGHTS(R)-WSU than in older versions which took two values, Astart and Astop, to ramp between them. This functionality is now available in a more general form through the fix adapt command, as explained below. Note that if you use an old input script and specify Astart and Astop without a cutoff, then LIGGGHTS(R)-WSU will interpret that as A and a cutoff, which is probabably not what you want.

The fix adapt command can be used to vary A for one or more pair types over the course of a simulation, in which case pair_coeff settings for A must still be specified, but will be overridden. For example these commands will vary the prefactor A for all pairwise interactions from 0.0 at the beginning to 30.0 at the end of a run:

```
variable prefactor equal ramp(0,30)
fix 1 all adapt 1 pair soft a * * v_prefactor
```

Note that a formula defined by an [equal-style variable](#) can use the current timestep, elapsed time in the current run, elapsed time since the beginning of a series of runs, as well as access other variables.

Mixing, shift, table, tail correction, restart, rRESPA info:

For atom type pairs I,J and I != J, the A coefficient and cutoff distance for this pair style can be mixed. A is always mixed via a *geometric* rule. The cutoff is mixed according to the pair_modify mix value. The default mix value is *geometric*. See the "pair_modify" command for details.

This pair style does not support the [pair_modify](#) shift option, since the pair interaction goes to 0.0 at the cutoff.

The [pair_modify](#) table and tail options are not relevant for this pair style.

This pair style writes its information to [binary restart files](#), so pair_style and pair_coeff commands do not need to be specified in an input script that reads a restart file.

This pair style can only be used via the *pair* keyword of the [run_style respa](#) command. It does not support the *inner*, *middle*, *outer* keywords.

Restrictions: none

Related commands:

[pair_coeff](#), [fix nve/limit](#), [fix adapt](#)

Default: none

pair_style sph/artVisc/tensCorr command

Syntax:

```
pair_style sph/artVisc/tensCorr kernelstyle args keyword values ...
```

- sph/artVisc/tensCorr = name of this pair_style command
- kernelstyle = *cubicspline* or *wendland*
- args = list of arguments for a particular style

```
cubicspline or wendland args = h
    h = smoothing length
```

- zero or more keyword/value pairs may be appended to args
- keyword = *artVisc* or *tensCorr*

```
artVisc values = alpha beta eta
    alpha = free parameter to control shear viscosity
    beta = free parameter to control bulk viscosity
    eta = coefficient to avoid singularities
  tensCorr values = epsilon deltap
    epsilon = free parameter
    deltap = initial particle distribution
```

Examples:

```
pair_style sph/artVisc/tensCorr wendland 0.001 artVisc 1e-4 0 1e-8
pair_style sph/artVisc/tensCorr cubicspline 0.001 artVisc 1e-4 0 1e-8 tensCorr 0.2 1e-2
```

Description:

The *sph/artVisc/tensCorr* style uses the smoothed particle hydrodynamics (SPH) method according to Monaghan [(1992)](#). The acting force is calculated from the acceleration as stated in the equation:

$$\frac{d\vec{v}_a}{dt} = -\sum_b m_b \left(\frac{P_a}{\rho_a^2} + \frac{P_b}{\rho_b^2} \right) \vec{\nabla}_a W_{ab}$$

Whereas the indices a and b stand for particles, P_j stands for pressure and ρ_j for the density. W_{ab} represents the kernel, which is defined by the kernelstyle.

For kernelstyle cubicspline a piecewise defined, 3-order kernel is used:

$$W(r,h) = \frac{1}{\pi h^3} \begin{cases} 1 - \frac{3}{2}q^2 + \frac{3}{4}q^3 & 0 \leq q < 1 \\ \frac{1}{4}(2-q)^3 & 1 \leq q < 2 \\ 0 & 2 \leq q \end{cases}$$

The wendland kernel (Wendland,1995) is defined as

$$W\left(r,h\right)=\frac{1}{\pi h^3}\left(1-\frac{q}{2}\right)^4\left(2q+1\right)\qquad 0\leq q\leq 2$$

The smoothing length h is the most important parameter for SPH-calculations. It depends on initial particle spacing, initial density ρ_0 and mass per particle m_j. In case that the smoothing length is about 1.2 times the initial particle spacing and it is a 3-dimensional cubic lattice (therefore the summation is over 57 particles), you can use the following equation [(Liu and Liu, 2003, p. 211-213)](#):

$$h_i^0=\frac{3}{32\pi}^{1/3}\left(\frac{\sum_{j=1}^{57}m_j}{\rho_i^0}\right)$$

The atom style *sph/var* uses the input argument h as initial smoothing length for all particles. In case the atom style *sph* (per-type smoothing length) is used an additional per-type property *sl* must be defined, e.g.,

```
fix          m2 all property/global sl peratomtype 0.0012
```

For further details on the basics of the SPH-method we recommend the papers from Monaghan [(1992)](#), [(1994)](#), etc.

Optionally, this pairstyle can take into account the artificial viscosity proposed my Monaghan (1985), if the artVisc keyword is appended. In this case, Π_{ab} is added to the bracket term in the above acceleration equation, where Π_{ab} is given by

$$\Pi_{ab}=\begin{cases}\frac{-\alpha\bar{c}_{ab}\mu_{ab}+\beta\mu_{ab}^2}{\bar{\rho}_{ab}}&\vec{v}_{ab}\cdot\vec{r}_{ab}<0\\0&\vec{v}_{ab}\cdot\vec{r}_{ab}\geq 0\end{cases}$$

and

$$\mu_{ab}=\frac{h\vec{v}_{ab}\cdot\vec{r}_{ab}}{\vec{r}_{ab}^2+\eta^2}$$

This expression produces a shear and bulk viscosity. The quadratic term enables simulation of high Mach number shocks. The parameter η^2 prevents singularities. A good choice is normally $\eta^2=0.01h^2$.

The choice of α and β should not be critical, although there are some aspects which you should take into account:

"In the present case, with negligible changes in the density [weakly compressible SPH], the viscosity is almost entirely shear viscosity with a viscosity coefficient approximately αhc." [(Monaghan, 1994)](#)

Bar-parameters like c_{ab} and ρ_{ab} are mean values of particle a and b. NOTE: μ_{ab} is calculated, and for the calculation of c_{ab} the per-type property *speedOfSound* has to be defined, e.g.,

```
fix              m1 all property/global speedOfSound peratomtype 20.
```

By appending the keyword *tensCorr* you enable the tensile correction algorithm [(Monaghan, 2000)](#) which improves results in combination with negative pressures (e.g. EOS like Tait's equation). This method adds $R*(f_{ab})^n$ to the bracket term, where the factor R is related to the pressure and can be calculated by $R=R_a+R_b$. In case of negative pressures ($P_a<0$) we use the rule

$$R_a = \frac{\epsilon |P_a|}{\rho_a^2}$$

otherwise R_a is zero. Typical values of *epsilon* are about 0.2. fab is calculated by

$$f_{ab} = \frac{W(r_{ab})}{W(\Delta p)}$$

where Δp denotes the initial particle spacing. NOTE: In a next version this calculation should be improved too.

Mixing, shift, table, tail correction, restart, rRESPA info:

The [pair_modify](#) mix, shift, table, and tail options are not relevant for sph pair styles.

These pair styles write their information to [binary restart files](#), so a pair_style command does not need to be specified in an input script that reads a restart file.

These pair styles can only be used via the *pair* keyword of the [run_style respa](#) command. They do not support the *inner*, *middle*, *outer* keywords.

Restrictions:

...

Related commands:

[pair_coeff](#)

Default: none

(**Liu and Liu, 2003**) "Smoothed Particle Hydrodynamics: A Meshfree Particle Method", G. R. Liu and M. B. Liu, World Scientific, p. 449 (2003).

(**Monaghan, 1992**) "Smoothed Particle Hydrodynamics", J. J. Monaghan, Annu. Rev. Astron. Astrophys., 30, p. 543-574 (1992).

(**Monaghan, 1994**) J. J. Monaghan, Journal of Computational Physics, 110, p. 399-406 (1994).

(**Monaghan, 2000**) J. J. Monaghan, Journal of Computational Physics, 159, p. 290-311 (2000).

pair_style sph command

Syntax:

pair_style sph kernelstyle args keyword values ...

- sph = name of this pair_style command
- kernelstyle = *cubicspline* or *wendland*
- args = list of arguments for a particular style

```
cubicspline or wendland args = h
    h = smoothing length
```
- zero or more keyword/value pairs may be appended to args
- keyword = *artVisc* or *tensCorr*

```
artVisc values = alpha beta cAB eta
    alpha = free parameter to control shear viscosity
    beta = free parameter to control bulk viscosity
    cAB = average speed of sound (velocity units)
    eta = coefficient to avoid singularities
tensCorr values = epsilon
    epsilon = free parameter
```

Examples:

```
pair_style sph wendland 0.001 artVisc 1e-4 0 1484. 1e-8
pair_style sph cubicspline 0.001 artVisc 1e-4 0 480 1e-8 tensCorr 0.2
```

LIGGGHTS vs. LAMMPS Info:

This command is not available in LAMMPS.

Description:

The *sph* style use the smoothed particle hydrodynamics (SPH) method according to Monaghan: "Smoothed Particle Hydrodynamics", J. J. Monaghan, Annual Review of Astronomy and Astrophysics (1992), Volume 30, Pages 543-574. The acting force is calculated from the acceleration as stated in the equation:

$$\frac{d\vec{v}_a}{dt} = -\sum_b m_b \left(\frac{P_b}{\rho_b^2} + \frac{P_a}{\rho_a^2} \right) \vec{\nabla}_a W_{ab}$$

Whereas the indices a and b stand for particles, P_j stands for pressure and ρ_j for the density. W_{ab} represents the kernel, which is defined by the kernelstyle.

For kernelstyle *cubicspline* a piecewise defined, 3-order kernel is used:

$$W(r,h) = \frac{1}{\pi h^3} \begin{cases} 1 - \frac{3}{2}q^2 + \frac{3}{4}q^3 & 0 \le q \le 1 \\ \frac{1}{4}(2-q)^3 & 1 \le q \le 2 \\ 0 & q \ge 2 \end{cases}$$

The *wendland* kernel (Wendland,1995) is defined as

$$W(r,h) = \frac{21}{16\pi h^3}\left(1 - \frac{q}{2}\right)^4 (2q + 1) \qquad 0 \le q \le 2$$

The smoothing length h is the most important parameter for SPH-calculations. It depends on initial particle spacing, initial density $_0$ and mass per particle m_j. In case that the smoothing length is about 1.2 times the initial particle spacing and it is a 3-dimensional cubic lattice (therefore the summation is over 57 particles), you can use the following equation (Liu and Liu, 2003, p. 211-213):

$$h_i^0 = \frac{3}{32\pi}^{1/3}\left(\frac{\sum_{j=1}^{57} m_j}{\rho_i^0}\right)$$

For further details on the basics of the SPH-method we recommend the papers from Monaghan 1992,1994, etc.

Optionally, this pairstyle can take into account the artificial viscosity proposed my Monaghan (1985), if the *artVisc* keyword is appended. In this case, $_{ab}$ is added to the bracket term in the above acceleration equation, where $_{ab}$ is given by

$$\Pi_{ab} = \begin{cases} \frac{-\alpha \bar{c}_{ab}\mu_{ab} + \beta \mu_{ab}^2}{\bar{\rho}_{ab}} & \vec{v}_{ab} \cdot \vec{r}_{ab} < 0 \\ 0 & \vec{v}_{ab} \cdot \vec{r}_{ab} > 0 \end{cases}$$

and

$$\mu_{ab} = \frac{h\vec{v}_{ab} \cdot \vec{r}_{ab}}{\vec{r}_{ab}^2 + \eta^2}.$$

This expression produces a shear and bulk viscosity. The quadratic term enables simulation of high Mach number shocks. The parameter ^{2} prevents singularities. A good choice is normally $^2 = 0.01h^2$. The choice of   and   should not be critical, although there are some aspects which you should take into account:

```
   In case of negligible changes in the density (weakly compressible SPH) it is almost only shear
```

Bar-parameters like c_{ab} and $_{ab}$ are mean values of particle a and b.
NOTE: $_{ab}$ is calculated, whereas c_{ab} is a given parameter. In a next version speed of sound should be added as atom property.

By appending the keyword *tensCorr* you enable the tensile correction algorithm (Monaghan, 2000) which improves results in combination with negative pressures (e.g. EOS like Tait's equation). This method adds $R*(f_{ab})^n$ to the bracket term, where the factor R is related to the pressure and can be calculated by $R = R_a + R_b$. In case of negative pressures ($P_a < 0$) we use the rule

$$R_a = \frac{\epsilon |P_a|}{\rho_a^2}$$

otherwise R_a is zero. Typical values of *epsilon* are about 0.2.
f_{ab} is calculated by

$$f_{ab} = \frac{W(r_{ab})}{W(\Delta p)}$$

where Δp denotes the initial particle spacing.
NOTE: In a next version this calculation should be improved too.

Restart info:

Until now there is no restart option implemented.

Restrictions:

...

Related commands:

pair_coeff

Default: none

(Monaghan1992) J. J. Monaghan, Annu. Rev. Astron. Astrophys., 30, p. 543-574 (1992).

(Monaghan1994) J. J. Monaghan, Journal of Computational Physics, 110, p. 399-406 (1994).

(Monaghan2000) J. J. Monaghan, Journal of Computational Physics, 159, p. 290-311 (2000).

pair_style command

Syntax:

```
pair_style style args
```

- style = one of the styles from the list [here](#)
- args = arguments used by a particular style

Examples:

```
pair_style gran model hertz tangential history
pair_style none
```

Description:

Set the formula(s) LIGGGHTS(R)-WSU uses to compute pairwise interactions. In LIGGGHTS(R)-WSU, pair potentials are defined between pairs of atoms that are within a cutoff distance and the set of active interactions typically changes over time. See the [bond_style](#) command to define potentials between pairs of bonded atoms, which typically remain in place for the duration of a simulation.

The full list of pair styles defined in LIGGGHTS(R)-WSU is on [this page](#).

Hybrid models where specified pairs of atom types interact via different pair potentials can be setup using the *hybrid* pair style.

The coefficients associated with a pair style are typically set for each pair of atom types, and are specified by the [pair_coeff](#) command or read from a file by the [read_data](#) or [read_restart](#) commands.

The [pair_modify](#) command sets options for mixing of type I-J interaction coefficients.

If the pair_style command has a cutoff argument, it sets global cutoffs for all pairs of atom types. The distance(s) can be smaller or larger than the dimensions of the simulation box.

Typically, the global cutoff value can be overridden for a specific pair of atom types by the [pair_coeff](#) command. The pair style settings (including global cutoffs) can be changed by a subsequent pair_style command using the same style. This will reset the cutoffs for all atom type pairs, including those previously set explicitly by a [pair_coeff](#) command. The exceptions to this are that pair_style *table* and *hybrid* settings cannot be reset. A new pair_style command for these styles will wipe out all previously specified pair_coeff values.

The full list of pair styles defined in LIGGGHTS(R)-WSU is on [this page](#).

Restrictions:

This command must be used before any coefficients are set by the [pair_coeff](#), [read_data](#), or [read_restart](#) commands.

Some pair styles are part of specific packages. They are only enabled if LIGGGHTS(R)-WSU was built with that package. See the [Making LIGGGHTS(R)-WSU](#) section for more info on packages. The doc pages for individual pair potentials tell if it is part of a package.

Related commands:

[pair_coeff](), [read_data](), [pair_modify](), [dielectric](),

Default:

```
pair_style none
```

partition command

Syntax:

```
partition style N command ...
```

- style = *yes* or *no*
- N = partition number (see asterisk form below)
- command = any LIGGGHTS(R)-WSU command

Examples:

```
partition yes 1 processors 4 10 6
partition no 5 print "Active partition"
partition yes *5 fix all nve
```

Description:

This command invokes the specified command on a subset of the partitions of processors you have defined via the -partition command-line switch. See [Section_start 6](#) for an explanation of the switch.

Normally, every input script command in your script is invoked by every partition. This behavior can be modified by defining world- or universe-style [variables](#) that have different values for each partition. This mechanism can be used to cause your script to jump to different input script files on different partitions, if such a variable is used in a [jump](#) command.

The "partition" command is another mechanism for having as input script operate differently on different partitions. It is basically a prefix on any LIGGGHTS(R)-WSU command. The commmand will only be invoked on the partition(s) specified by the *style* and *N* arguments.

If the *style* is *yes*, the command will be invoked on any partition which matches the *N* argument. If the *style* is *no* the command will be invoked on all the partitions which do not match the Np argument.

Partitions are numbered from 1 to Np, where Np is the number of partitions specified by the [-partition command-line switch](#).

N can be specified in one of two ways. An explicit numeric value can be used, as in the 1st example above. Or a wild-card asterisk can be used to span a range of partition numbers. This takes the form "*" or "*n" or "n*" or "m*n". An asterisk with no numeric values means all partitions from 1 to Np. A leading asterisk means all partitions from 1 to n (inclusive). A trailing asterisk means all partitions from n to Np (inclusive). A middle asterisk means all partitions from m to n (inclusive).

Restrictions: none

Related commands: none

Default: none

print command

Syntax:

print string keyword value:pre

- string = text string to print, which may contain variables
- zero or more keyword/value pairs may be appended
- keyword = *file* or *append* or *screen*

```
  file value = filename
    append value = filename
    screen value = yes or no
```

Examples:

```
print "Done with equilibration" file info.dat
print Vol=$v append info.dat screen no
print "The system volume is now $v"
print 'The system volume is now $v'
```

Description:

Print a text string to the screen and logfile. One line of output is generated. The text string must be a single argument, so it should be enclosed in double quotes if it is more than one word. If it contains variables, they will be evaluated and their current values printed.

If the *file* or *append* keyword is used, a filename is specified to which the output will be written. If *file* is used, then the filename is overwritten if it already exists. If *append* is used, then the filename is appended to if it already exists, or created if it does not exist.

If the *screen* keyword is used, output to the screen and logfile can be turned on or off as desired.

If you want the print command to be executed multiple times (with changing variable values), there are 3 options. First, consider using the [fix print](#) command, which will print a string periodically during a simulation. Second, the print command can be used as an argument to the *every* option of the [run](#) command. Third, the print command could appear in a section of the input script that is looped over (see the [jump](#) and [next](#) commands).

See the [variable](#) command for a description of *equal* style variables which are typically the most useful ones to use with the print command. Equal-style variables can calculate formulas involving mathematical operations, atom properties, group properties, thermodynamic properties, global values calculated by a [compute](#) or [fix](#), or references to other [variables](#).

Restrictions: none

Related commands:

[fix print](#), [variable](#)

Default:

The option defaults are no file output and screen = yes.

processors command

Syntax:

```
processors Px Py Pz keyword args ...
```

- Px,Py,Pz = # of processors in each dimension of 3d grid overlaying the simulation domain
- zero or more keyword/arg pairs may be appended
- keyword = *grid* or *map* or *part* or *file*

```
  grid arg = gstyle params ...
      gstyle = onelevel or twolevel or numa or custom
        onelevel params = none
        twolevel params = Nc Cx Cy Cz
          Nc = number of cores per node
          Cx,Cy,Cz = # of cores in each dimension of 3d sub-grid assigned to each node
        numa params = none
        custom params = infile
          infile = file containing grid layout
    map arg = cart or cart/reorder or xyz or xzy or yxz or yzx or zxy or zyx
      cart = use MPI_Cart() methods to map processors to 3d grid with reorder = 0
      cart/reorder = use MPI_Cart() methods to map processors to 3d grid with reorder = 1
      xyz,xzy,yxz,yzx,zxy,zyx = map procesors to 3d grid in IJK ordering
  numa arg = none
  part args = Psend Precv cstyle
    Psend = partition # (1 to Np) which will send its processor layout
    Precv = partition # (1 to Np) which will recv the processor layout
    cstyle = multiple
      multiple = Psend grid will be multiple of Precv grid in each dimension
  file arg = outfile
    outfile = name of file to write 3d grid of processors to
```

Examples:

```
processors * * 5
processors 2 4 4
processors * * 8 map xyz
processors * * * grid numa
processors * * * grid twolevel 4 * * 1
processors 4 8 16 grid custom myfile
processors * * * part 1 2 multiple
```

Description:

Specify how processors are mapped as a 3d logical grid to the global simulation box. This involves 2 steps. First if there are P processors it means choosing a factorization P = Px by Py by Pz so that there are Px processors in the x dimension, and similarly for the y and z dimensions. Second, the P processors are mapped to the logical 3d grid. The arguments to this command control each of these 2 steps.

The Px, Py, Pz parameters affect the factorization. Any of the 3 parameters can be specified with an asterisk "*", which means LIGGGHTS(R)-WSU will choose the number of processors in that dimension of the grid. It will do this based on the size and shape of the global simulation box so as to minimize the surface-to-volume ratio of each processor's sub-domain.

Since LIGGGHTS(R)-WSU does not load-balance by changing the grid of 3d processors on-the-fly, choosing explicit values for Px or Py or Pz can be used to override the LIGGGHTS(R)-WSU default if it is known to be

sub-optimal for a particular problem. E.g. a problem where the extent of atoms will change dramatically in a particular dimension over the course of the simulation.

The product of Px, Py, Pz must equal P, the total # of processors LIGGGHTS(R)-WSU is running on. For a 2d simulation, Pz must equal 1.

Note that if you run on a prime number of processors P, then a grid such as 1 x P x 1 will be required, which may incur extra communication costs due to the high surface area of each processor's sub-domain.

Also note that 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. Also note that you can prefix the processors command with the partition command to easily specify different Px,Py,Pz values for different partitions.

You can use the partition command to specify different processor grids for different partitions, e.g.

```
partition yes 1 processors 4 4 4
partition yes 2 processors 2 3 2
```

The *grid* keyword affects the factorization of P into Px,Py,Pz and it can also affect how the P processor IDs are mapped to the 3d grid of processors.

The *onelevel* style creates a 3d grid that is compatible with the Px,Py,Pz settings, and which minimizes the surface-to-volume ratio of each processor's sub-domain, as described above. The mapping of processors to the grid is determined by the *map* keyword setting.

The *twolevel* style can be used on machines with multicore nodes to minimize off-node communication. It insures that contiguous sub-sections of the 3d grid are assigned to all the cores of a node. For example if *Nc* is 4, then 2x2x1 or 2x1x2 or 1x2x2 sub-sections of the 3d grid will correspond to the cores of each node. This affects both the factorization and mapping steps.

The *Cx*, *Cy*, *Cz* settings are similar to the *Px*, *Py*, *Pz* settings, only their product should equal *Nc*. Any of the 3 parameters can be specified with an asterisk "*", which means LIGGGHTS(R)-WSU will choose the number of cores in that dimension of the node's sub-grid. As with Px,Py,Pz, it will do this based on the size and shape of the global simulation box so as to minimize the surface-to-volume ratio of each processor's sub-domain.

IMPORTANT NOTE: For the *twolevel* style to work correctly, it assumes the MPI ranks of processors LIGGGHTS(R)-WSU is running on are ordered by core and then by node. E.g. if you are running on 2 quad-core nodes, for a total of 8 processors, then it assumes processors 0,1,2,3 are on node 1, and processors 4,5,6,7 are on node 2. This is the default rank ordering for most MPI implementations, but some MPIs provide options for this ordering, e.g. via environment variable settings.

The *numa* style operates similar to the *twolevel* keyword except that it auto-detects which cores are running on which nodes. Currently, it does this in only 2 levels, but it may be extended in the future to account for socket topology and other non-uniform memory access (NUMA) costs. It also uses a different algorithm than the *twolevel* keyword for doing the two-level factorization of the simulation box into a 3d processor grid to minimize off-node communication, and it does its own MPI-based mapping of nodes and cores to the logical 3d grid. Thus it may produce a different layout of the processors than the *twolevel* options.

The *numa* style will give an error if the number of MPI processes is not divisible by the number of cores used per node, or any of the Px or Py of Pz values is greater than 1.

IMPORTANT NOTE: Unlike the *twolevel* style, the *numa* style does not require any particular ordering of MPI ranks i norder to work correctly. This is because it auto-detects which processes are running on which nodes.

The *custom* style uses the file *infile* to define both the 3d factorization and the mapping of processors to the grid.

The file should have the following format. Any number of initial blank or comment lines (starting with a "#" character) can be present. The first non-blank, non-comment line should have 3 values:

```
Px Py Py
```

These must be compatible with the total number of processors and the Px, Py, Pz settings of the processors commmand.

This line should be immediately followed by P = Px*Py*Pz lines of the form:

```
ID I J K
```

where ID is a processor ID (from 0 to P-1) and I,J,K are the processors location in the 3d grid. I must be a number from 1 to Px (inclusive) and similarly for J and K. The P lines can be listed in any order, but no processor ID should appear more than once.

The *map* keyword affects how the P processor IDs (from 0 to P-1) are mapped to the 3d grid of processors. It is only used by the *onelevel* and *twolevel* grid settings.

The *cart* style uses the family of MPI Cartesian functions to perform the mapping, namely MPI_Cart_create(), MPI_Cart_get(), MPI_Cart_shift(), and MPI_Cart_rank(). It invokes the MPI_Cart_create() function with its reorder flag = 0, so that MPI is not free to reorder the processors.

The *cart/reorder* style does the same thing as the *cart* style except it sets the reorder flag to 1, so that MPI can reorder processors if it desires.

The *xyz*, *xzy*, *yxz*, *yzx*, *zxy*, and *zyx* styles are all similar. If the style is IJK, then it maps the P processors to the grid so that the processor ID in the I direction varies fastest, the processor ID in the J direction varies next fastest, and the processor ID in the K direction varies slowest. For example, if you select style *xyz* and you have a 2x2x2 grid of 8 processors, the assignments of the 8 octants of the simulation domain will be:

```
proc 0 = lo x, lo y, lo z octant
proc 1 = hi x, lo y, lo z octant
proc 2 = lo x, hi y, lo z octant
proc 3 = hi x, hi y, lo z octant
proc 4 = lo x, lo y, hi z octant
proc 5 = hi x, lo y, hi z octant
proc 6 = lo x, hi y, hi z octant
proc 7 = hi x, hi y, hi z octant
```

Note that, in principle, an MPI implementation on a particular machine should be aware of both the machine's network topology and the specific subset of processors and nodes that were assigned to your simulation. Thus its MPI_Cart calls can optimize the assignment of MPI processes to the 3d grid to minimize communication costs. In practice, however, few if any MPI implementations actually do this. So it is likely that the *cart* and *cart/reorder* styles simply give the same result as one of the IJK styles.

Also note, that for the *twolevel* grid style, the *map* setting is used to first map the nodes to the 3d grid, then again to the cores within each node. For the latter step, the *cart* and *cart/reorder* styles are not supported, so an *xyz* style is used in their place.

The *part* keyword affects the factorization of P into Px,Py,Pz.

processors command

It can be useful when running in multi-partition mode, e.g. with the <u>run_style verlet/split</u> command. It specifies a dependency bewteen a sending partition *Psend* and a receiving partition *Precv* which is enforced when each is setting up their own mapping of their processors to the simulation box. Each of *Psend* and *Precv* must be integers from 1 to Np, where Np is the number of partitions you have defined via the <u>-partition command-line switch</u>.

A "dependency" means that the sending partition will create its 3d logical grid as Px by Py by Pz and after it has done this, it will send the Px,Py,Pz values to the receiving partition. The receiving partition will wait to receive these values before creating its own 3d logical grid and will use the sender's Px,Py,Pz values as a constraint. The nature of the constraint is determined by the *cstyle* argument.

For a *cstyle* of *multiple*, each dimension of the sender's processor grid is required to be an integer multiple of the corresponding dimension in the receiver's processor grid. This is a requirement of the <u>run_style verlet/split</u> command.

For example, assume the sending partition creates a 4x6x10 grid = 240 processor grid. If the receiving partition is running on 80 processors, it could create a 4x2x10 grid, but it will not create a 2x4x10 grid, since in the y-dimension, 6 is not an integer multiple of 4.

IMPORTANT NOTE: If you use the <u>partition</u> command to invoke different "processsors" commands on different partitions, and you also use the *part* keyword, then you must insure that both the sending and receiving partitions invoke the "processors" command that connects the 2 partitions via the *part* keyword. LIGGGHTS(R)-WSU cannot easily check for this, but your simulation will likely hang in its setup phase if this error has been made.

The *file* keyword writes the mapping of the factorization of P processors and their mapping to the 3d grid to the specified file *outfile*. This is useful to check that you assigned physical processors in the manner you desired, which can be tricky to figure out, especially when running on multiple partitions or on, a multicore machine or when the processor ranks were reordered by use of the <u>-reorder command-line switch</u> or due to use of MPI-specific launch options such as a config file.

If you have multiple partitions you should insure that each one writes to a different file, e.g. using a <u>world-style variable</u> for the filename. The file has a self-explanatory header, followed by one-line per processor in this format:

world-ID universe-ID original-ID: I J K: name

The IDs are the processor's rank in this simulation (the world), the universe (of multiple simulations), and the original MPI communicator used to instantiate LIGGGHTS(R)-WSU, respectively. The world and universe IDs will only be different if you are running on more than one partition; see the <u>-partition command-line switch</u>. The universe and original IDs will only be different if you used the <u>-reorder command-line switch</u> to reorder the processors differently than their rank in the original communicator LIGGGHTS(R)-WSU was instantiated with.

I,J,K are the indices of the processor in the 3d logical grid, each from 1 to Nd, where Nd is the number of processors in that dimension of the grid.

The *name* is what is returned by a call to MPI_Get_processor_name() and should represent an identifier relevant to the physical processors in your machine. Note that depending on the MPI implementation, multiple cores can have the same *name*.

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.

The *grid numa* keyword only currently works with the *map cart* option.

The *part* keyword (for the receiving partition) only works with the *grid onelevel* or *grid twolevel* options.

Related commands:

partition, [-reorder command-line switch](-reorder)

Default:

The option defaults are Px Py Pz = * * *, grid = onelevel, and map = cart.

quit command

Syntax:

```
quit
```

Examples:

```
quit
if "$n > 10000" then quit
```

Description:

This command causes LIGGGHTS(R)-WSU to exit, after shutting down all output cleanly.

It can be used as a debug statement in an input script, to terminate the script at some intermediate point.

It can also be used as an invoked command inside the "then" or "else" portion of an if command.

Restrictions: none

Related commands:

if

Default: none

read_data command

Syntax:

```
read_data file keyword args ...
```

- file = name of data file to read in
- zero or more keyword/arg pairs may be appended
- keyword = *fix*

```
  fix args = fix-ID header-string section-string
      fix-ID = ID of fix to process header lines and sections of data file
      header-string = header lines containing this string will be passed to fix
      section-string = section names with this string will be passed to fix
```

Examples:

```
read_data data.lj
read_data ../run7/data.polymer.gz
read_data data.protein fix mycmap crossterm CMAP
```

Description:

Read in a data file containing information LIGGGHTS(R)-WSU needs to run a simulation. The file can be ASCII text or a gzipped text file (detected by a .gz suffix). This is one of 3 ways to specify initial atom coordinates; see the [read_restart](#) and [create_atoms](#) commands for alternative methods.

The structure of the data file is important, though many settings and sections are optional or can come in any order. See the examples directory for sample data files for different problems.

A data file has a header and a body. The header appears first. The first line of the header is always skipped; it typically contains a description of the file. Then lines are read one at a time. Lines can have a trailing comment starting with '#' that is ignored. If the line is blank (only whitespace after comment is deleted), it is skipped. If the line contains a header keyword, the corresponding value(s) is read from the line. If it doesn't contain a header keyword, the line begins the body of the file.

The body of the file contains zero or more sections. The first line of a section has only a keyword. The next line is skipped. The remaining lines of the section contain values. The number of lines depends on the section keyword as described below. Zero or more blank lines can be used between sections. Sections can appear in any order, with a few exceptions as noted below.

The keyword *fix* can be used one or more times. Each usage specifies a fix that will be used to process a specific portion of the data file. Any header line containing *header-string* and any section with a name containing *section-string* will be passed to the specified fix. See the [fix property/atom](#) command for an example of a fix that operates in this manner. The doc page for the fix defines the syntax of the header line(s) and section(s) that it reads from the data file. Note that the *header-string* can be specified as NULL, in which case no header lines are passed to the fix. This means that it can infer the length of its Section from standard header settings, such as the number of atoms.

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 the 2 words in "xlo xhi" or the 2 words in "Bond Coeffs", is not valid.

These are the recognized header keywords. Header lines can come in any order. The value(s) are read from the beginning of the line. Thus the keyword *atoms* should be in a line like "1000 atoms"; the keyword *ylo yhi* should be in a line like "-10.0 10.0 ylo yhi"; the keyword *xy xz yz* should be in a line like "0.0 5.0 6.0 xy xz yz". All these settings have a default value of 0, except the lo/hi box size defaults are -0.5 and 0.5. A line need only appear if the value is different than the default.

- *atoms* = # of atoms in system
- *bonds* = # of bonds in system
- *atom types* = # of atom types in system
- *bond types* = # of bond types in system
- *extra bond per atom* = leave space for this many new bonds per atom
- *ellipsoids* = # of ellipsoids in system
- *lines* = # of line segments in system
- *xlo xhi* = simulation box boundaries in x dimension
- *ylo yhi* = simulation box boundaries in y dimension
- *zlo zhi* = simulation box boundaries in z dimension
- *xy xz yz* = simulation box tilt factors for triclinic system

The initial simulation box size is determined by the lo/hi settings. In any dimension, the system may be periodic or non-periodic; see the boundary command.

If the *xy xz yz* line does not appear, LIGGGHTS(R)-WSU will set up an axis-aligned (orthogonal) simulation box. If the line does appear, LIGGGHTS(R)-WSU creates a non-orthogonal simulation domain shaped as a parallelepiped with triclinic symmetry. The parallelepiped has its "origin" at (xlo,ylo,zlo) and is defined by 3 edge vectors starting from the origin given by A = (xhi-xlo,0,0); B = (xy,yhi-ylo,0); C = (xz,yz,zhi-zlo). *Xy,xz,yz* can be 0.0 or positive or negative values and are called "tilt factors" because they are the amount of displacement applied to faces of an originally orthogonal box to transform it into the parallelepiped.

The tilt factors (xy,xz,yz) can not skew the box more than half the distance of the corresponding parallel box length. For example, if xlo = 2 and xhi = 12, then the x box length is 10 and the xy tilt factor must be between -5 and 5. Similarly, both xz and yz must be between -(xhi-xlo)/2 and +(yhi-ylo)/2. Note that this is not a limitation, since if the maximum tilt factor is 5 (as in this example), then configurations with tilt = ..., -15, -5, 5, 15, 25, ... are all geometrically equivalent.

See Section_howto 12 of the doc pages for a geometric description of triclinic boxes, as defined by LIGGGHTS(R)-WSU, and how to transform these parameters to and from other commonly used triclinic representations.

When a triclinic system is used, the simulation domain must be periodic in any dimensions with a non-zero tilt factor, as defined by the boundary command. I.e. if the xy tilt factor is non-zero, then both the x and y dimensions must be periodic. Similarly, x and z must be periodic if xz is non-zero and y and z must be periodic if yz is non-zero. Also note that if your simulation will tilt the box, e.g. via the fix deform command, the simulation box must be defined as triclinic, even if the tilt factors are initially 0.0.

For 2d simulations, the *zlo zhi* values should be set to bound the z coords for atoms that appear in the file; the default of -0.5 0.5 is valid if all z coords are 0.0. For 2d triclinic simulations, the xz and yz tilt factors must be 0.0.

If the system is periodic (in a dimension), then atom coordinates can be outside the bounds (in that dimension); they will be remapped (in a periodic sense) back inside the box.

IMPORTANT NOTE: If the system is non-periodic (in a dimension), then all atoms in the data file must have coordinates (in that dimension) that are "greater than or equal to" the lo value and "less than or equal to" the hi

value. If the non-periodic dimension is of style "fixed" (see the [boundary](#) command), then the atom coords must be strictly "less than" the hi value, due to the way LIGGGHTS(R)-WSU assign atoms to processors. Note that you should not make the lo/hi values radically smaller/larger than the extent of the atoms. For example, if your atoms extend from 0 to 50, you should not specify the box bounds as -10000 and 10000. This is because LIGGGHTS(R)-WSU uses the specified box size to layout the 3d grid of processors. A huge (mostly empty) box will be sub-optimal for performance when using "fixed" boundary conditions (see the [boundary](#) command). When using "shrink-wrap" boundary conditions (see the [boundary](#) command), a huge (mostly empty) box may cause a parallel simulation to lose atoms the first time that LIGGGHTS(R)-WSU shrink-wraps the box around the atoms.

The "extra bond per atom" setting should be used if new bonds will be added to the system when a simulation runs, e.g. by using the [fix bond/create](#) command. This will pre-allocate space in LIGGGHTS(R)-WSU data structures for storing the new bonds.

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

- *Atoms, Velocities, Masses, Ellipsoids, Lines* = atom-property sections
- *Bonds* = molecular topology sections
- *Pair Coeffs, PairIJ Coeffs, Bond Coeffs* = force field sections
- *BondBond Coeffs, BondAngle Coeffs, MiddleBondTorsion Coeffs, EndBondTorsion Coeffs* = class 2 force field sections

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

Any individual line in the various sections can have a trailing comment starting with "#" for annotation purposes. E.g. in the Atoms section:

```
10 1 17 -1.0 10.0 5.0 6.0    # salt ion
```

Atoms section:

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

An *Atoms* section must appear in the data file if natoms > 0 in the header section. The atoms can be listed in any order. These are the line formats for each [atom style](#) in LIGGGHTS(R)-WSU. As discussed below, each line can optionally have 3 flags (nx,ny,nz) appended to it, which indicate which image of a periodic simulation box the atom is in. These may be important to include for some kinds of analysis.

bond	atom-ID molecule-ID atom-type x y z
ellipsoid	atom-ID atom-type ellipsoidflag density x y z
line	atom-ID molecule-ID atom-type lineflag density x y z
molecular	atom-ID molecule-ID atom-type x y z
hybrid	atom-ID atom-type x y z sub-style1 sub-style2 ...

The keywords have these meanings:

- atom-ID = integer ID of atom
- molecule-ID = integer ID of molecule the atom belongs to
- atom-type = type of atom (1-Ntype)
- q = charge on atom (charge units)
- diameter = diameter of spherical atom (distance units)
- ellipsoidflag = 1 for ellipsoidal particles, 0 for point particles

- lineflag = 1 for line segment particles, 0 for point particles
- triangleflag = 1 for triangular particles, 0 for point particles
- density = density of particle (mass/distance^3 or mass/distance^2 or mass/distance units, depending on dimensionality of particle)
- mass = mass of particle (mass units)
- volume = volume of particle (distance^3 units)
- x,y,z = coordinates of atom

The units for these quantities depend on the unit style; see the units command for details.

For 2d simulations specify z as 0.0, or a value within the *zlo zhi* setting in the data file header.

The atom-ID is used to identify the atom throughout the simulation and in dump files. Normally, it is a unique value from 1 to Natoms for each atom. Unique values larger than Natoms can be used, but they will cause extra memory to be allocated on each processor, if an atom map array is used (see the atom_modify command). If an atom map array is not used (e.g. an atomic system with no bonds), and velocities are not assigned in the data file, and you don't care if unique atom IDs appear in dump files, then the atom-IDs can all be set to 0.

The molecule ID is a 2nd identifier attached to an atom. Normally, it is a number from 1 to N, identifying which molecule the atom belongs to. It can be 0 if it is an unbonded atom or if you don't care to keep track of molecule assignments.

The diameter specifies the size of a finite-size spherical particle. It can be set to 0.0, which means that atom is a point particle.

The ellipsoidflag, lineflag, triangleflag determine whether the particle is a finite-size ellipsoid or line or triangle or body of finite size, or whether the particle is a point particle. Additional attributes must be defined for each ellipsoid, line, triangle, or body in the corresponding *Ellipsoids*, *Lines*, *Triangles* section.

Some pair styles and fixes and computes that operate on finite-size particles allow for a mixture of finite-size and point particles. See the doc pages of individual commands for details.

For finite-size particles, the density is used in conjunction with the particle volume to set the mass of each particle as mass = density * volume. In this context, volume can be a 3d quantity (for spheres or ellipsoids), a 2d quantity (for triangles), or a 1d quantity (for line segments). If the volume is 0.0, meaning a point particle, then the density value is used as the mass. One exception is for the body atom style, in which case the mass of each particle (body or point particle) is specified explicitly. This is because the volume of the body is unknown.

For atom_style hybrid, following the 5 initial values (ID,type,x,y,z), specific values for each sub-style must be listed. The order of the sub-styles is the same as they were listed in the atom_style command. The sub-style specific values are those that are not the 5 standard ones (ID,type,x,y,z). For example, for the "charge" sub-style, a "q" value would appear. For the "full" sub-style, a "molecule-ID" and "q" would appear. These are listed in the same order they appear as listed above. Thus if

```
atom_style hybrid charge sphere
```

were used in the input script, each atom line would have these fields:

```
atom-ID atom-type x y z q diameter density
```

Note that if a non-standard value is defined by multiple sub-styles, it must appear mutliple times in the atom line. E.g. the atom line for atom_style hybrid dipole full would list "q" twice:

```
atom-ID atom-type x y z q mux muy myz molecule-ID q
```

Atom lines (all lines or none of them) can optionally list 3 trailing integer values: nx,ny,nz. For periodic dimensions, they specify which image of the simulation box the atom is considered to be in. An image of 0 means it is inside the box as defined. A value of 2 means add 2 box lengths to get the true value. A value of -1 means subtract 1 box length to get the true value. LIGGGHTS(R)-WSU updates these flags as atoms cross periodic boundaries during the simulation. The flags can be output with atom snapshots via the dump command.

If nx,ny,nz values are not set in the data file, LIGGGHTS(R)-WSU initializes them to 0. If image information is needed for later analysis and they are not all initially 0, it's important to set them correctly in the data file. Also, if you plan to use the replicate command to generate a larger system, these flags must be listed correctly for bonded atoms when the bond crosses a periodic boundary. I.e. the values of the image flags should be different by 1 (in the appropriate dimension) for the two atoms in such a bond.

Atom velocities and other atom quantities not defined above are set to 0.0 when the *Atoms* section is read. Velocities can be set later by a *Velocities* section in the data file or by a velocity or set command in the input script.

Bond Coeffs section:

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

    ```
    ID = bond type (1-N)
    coeffs = list of coeffs
    ```
- example:

    ```
    4 250 1.49
    ```

The number and meaning of the coefficients are specific to the defined bond style. See the bond_style and bond_coeff commands for details. Coefficients can also be set via the bond_coeff command in the input script.

BondBond Coeffs section:

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

    ```
    ID = angle type (1-N)
    coeffs = list of coeffs (see class 2 section of angle_coeff)
    ```

BondBond13 Coeffs section:

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

    ```
    ID = dihedral type (1-N)
    coeffs = list of coeffs (see class 2 section of dihedral_coeff)
    ```

Bonds section:

- one line per bond
- line syntax: ID type atom1 atom2

    ```
    ID = bond number (1-Nbonds)
    ```

read_data command   415

```
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).

Ellipsoids section:

- one line per ellipsoid
- line syntax: atom-ID shapex shapey shapez quatw quati quatj quatk

```
atom-ID = ID of atom which is an ellipsoid
shapex,shapey,shapez = 3 diameters of ellipsoid (distance units)
quatw,quati,quatj,quatk = quaternion components for orientation of atom
```
- example:

```
12 1 2 1 1 0 0 0
```

The *Ellipsoids* section must appear if [atom_style ellipsoid](#) is used and any atoms are listed in the *Atoms* section with an ellipsoidflag = 1. The number of ellipsoids should be specified in the header section via the "ellipsoids" keyword.

The 3 shape values specify the 3 diameters or aspect ratios of a finite-size ellipsoidal particle, when it is oriented along the 3 coordinate axes. They must all be non-zero values.

The values *quatw*, *quati*, *quatj*, and *quatk* set the orientation of the atom as a quaternion (4-vector). Note that the shape attributes specify the aspect ratios of an ellipsoidal particle, which is oriented by default with its x-axis along the simulation box's x-axis, and similarly for y and z. If this body is rotated (via the right-hand rule) by an angle theta around a unit vector (a,b,c), then the quaternion that represents its new orientation is given by (cos(theta/2), a*sin(theta/2), b*sin(theta/2), c*sin(theta/2)). These 4 components are quatw, quati, quatj, and quatk as specified above. LIGGGHTS(R)-WSU normalizes each atom's quaternion in case (a,b,c) is not specified as a unit vector.

The *Ellipsoids* section must appear after the *Atoms* section.

EndBondTorsion Coeffs section:

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

```
ID = dihedral type (1-N)
coeffs = list of coeffs (see class 2 section of dihedral coeff)
```

Lines section:

- one line per line segment
- line syntax: atom-ID x1 y1 x2 y2

```
atom-ID = ID of atom which is a line segment
x1,y1 = 1st end point
x2,y2 = 2nd end point
```
- example:

```
12 1.0 0.0 2.0 0.0
```

The *Lines* section must appear if [atom_style line](#) is used and any atoms are listed in the *Atoms* section with a lineflag = 1. The number of lines should be specified in the header section via the "lines" keyword.

The 2 end points are the end points of the line segment. The ordering of the 2 points should be such that using a right-hand rule to cross the line segment with a unit vector in the +z direction, gives an "outward" normal vector perpendicular to the line segment. I.e. normal = (c2-c1) x (0,0,1). This orientation may be important for defining some interactions.

The *Lines* section must appear after the *Atoms* section.

Masses section:

- one line per atom type
- line syntax: ID mass

```
ID = atom type (1-N)
mass = mass value
```

- example:

```
3 1.01
```

This defines the mass of each atom type. This can also be set via the [mass](#) command in the input script. This section cannot be used for atom styles that define a mass for individual atoms - e.g. [atom_style sphere](#).

MiddleBondTorsion Coeffs section:

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

```
ID = dihedral type (1-N)
coeffs = list of coeffs (see class 2 section of dihedral_coeff)
```

Pair Coeffs section:

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

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

- example:

```
3 0.022 2.35197 0.022 2.35197
```

The number and meaning of the coefficients are specific to the defined pair style. See the [pair_style](#) and [pair_coeff](#) commands for details. Since pair coefficients for types I != J are not specified, these will be generated automatically by the pair style's mixing rule. See the individual pair_style doc pages and the [pair_modify mix](#) command for details. Pair coefficients can also be set via the [pair_coeff](#) command in the input script.

PairIJ Coeffs section:

- one line per pair of atom types for all I,J with I <= J
- line syntax: ID1 ID2 coeffs

```
ID1 = atom type I = 1-N
ID2 = atom type J = I-N, with I <= J
```

```
coeffs = list of coeffs
```
  • examples:

```
3 3 0.022 2.35197 0.022 2.35197
3 5 0.022 2.35197 0.022 2.35197
```

This section must have N*(N+1)/2 lines where N = # of atom types. The number and meaning of the coefficients are specific to the defined pair style. See the pair_style and pair_coeff commands for details. Since pair coefficients for types I != J are all specified, these values will turn off the default mixing rule defined by the pair style. See the individual pair_style doc pages and the pair_modify mix command for details. Pair coefficients can also be set via the pair_coeff command in the input script.

Triangles section:

  • one line per triangle
  • line syntax: atom-ID x1 y1 x2 y2

```
atom-ID = ID of atom which is a line segment
x1,y1,z1 = 1st corner point
x2,y2,z2 = 2nd corner point
x3,y3,z3 = 3rd corner point
```
  • example:

```
12 0.0 0.0 0.0 2.0 0.0 1.0 0.0 2.0 1.0
```

The *Triangles* section must appear if atom_style tri is used and any atoms are listed in the *Atoms* section with a triangleflag = 1. The number of lines should be specified in the header section via the "triangles" keyword.

The 3 corner points are the corner points of the triangle. The ordering of the 3 points should be such that using a right-hand rule to go from point1 to point2 to point3 gives an "outward" normal vector to the face of the triangle. I.e. normal = (c2-c1) x (c3-c1). This orientation may be important for defining some interactions.

The *Triangles* section must appear after the *Atoms* section.

Velocities section:

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

all styles except those listed	atom-ID vx vy vz
electron	atom-ID vx vy vz ervel
ellipsoid	atom-ID vx vy vz lx ly lz
sphere	atom-ID vx vy vz wx wy wz
hybrid	atom-ID vx vy vz sub-style1 sub-style2 ...

where the keywords have these meanings:

vx,vy,vz = translational velocity of atom lx,ly,lz = angular momentum of aspherical atom wx,wy,wz = angular velocity of spherical atom ervel = electron radial velocity (0 for fixed-core):ul

The velocity lines can appear in any order. This section can only be used after an *Atoms* section. This is because the *Atoms* section must have assigned a unique atom ID to each atom so that velocities can be assigned to them.

Vx, vy, vz, and ervel are in units of velocity. Lx, ly, lz are in units of angular momentum (distance-velocity-mass). Wx, Wy, Wz are in units of angular velocity (radians/time).

read_data command

For atom_style hybrid, following the 4 initial values (ID,vx,vy,vz), specific values for each sub-style must be listed. The order of the sub-styles is the same as they were listed in the atom_style command. The sub-style specific values are those that are not the 5 standard ones (ID,vx,vy,vz). For example, for the "sphere" sub-style, "wx", "wy", "wz" values would appear. These are listed in the same order they appear as listed above. Thus if

```
atom_style hybrid electron sphere
```

were used in the input script, each velocity line would have these fields:

```
atom-ID vx vy vz ervel wx wy wz
```

Translational velocities can also be set by the velocity command in the input script.

Restrictions:

To read gzipped data files, you must compile LIGGGHTS(R)-WSU with the -DLAMMPS_GZIP option - see the Making LIGGGHTS(R)-WSU section of the documentation.

Related commands:

read_dump, read_restart, create_atoms, write_data

Default: none

read_dump command

Syntax:

```
read_dump file Nstep field1 field2 ... keyword values ...
```

- file = name of dump file to read
- Nstep = snapshot timestep to read from file
- one or more fields may be appended

```
field = x or y or z or vx or vy or vz or q or ix or iy or iz
  x,y,z = atom coordinates
  vx,vy,vz = velocity components
  q = charge
  ix,iy,iz = image flags in each dimension
```
- zero or more keyword/value pairs may be appended
- keyword = *box* or *replace* or *purge* or *trim* or *add* or *label* or *scaled* or *wrapped* or *format*

```
box value = yes or no = replace simulation box with dump box
  replace value = yes or no = overwrite atoms with dump atoms
  purge value = yes or no = delete all atoms before adding dump atoms
  trim value = yes or no = trim atoms not in dump snapshot
  add value = yes or no = add new dump atoms to system
  label value = field column
    field = one of the listed fields or id or type
    column = label on corresponding column in dump file
  scaled value = yes or no = coords in dump file are scaled/unscaled
  wrapped value = yes or no = coords in dump file are wrapped/unwrapped
  format values = format of dump file, must be last keyword if used
    native = native LIGGGHTS(R)-WSU dump file
    xyz = XYZ file
```

Examples:

```
read_dump dump.file 5000 x y z
read_dump dump.xyz 5 x y z format xyz box no
read_dump dump.file 5000 x y vx vy trim yes
read_dump ../run7/dump.file.gz 10000 x y z box yes
read_dump dump.xyz 5 x y z box no format xyz
```

Description:

Read atom information from a dump file to overwrite the current atom coordinates, and optionally the atom velocities and image flags and the simluation box dimensions. This is useful for restarting a run from a particular snapshot in a dump file. See the [read_restart](#) and [read_data](#) commands for alternative methods to do this. Also see the [rerun](#) command for a means of reading multiple snapshots from a dump file.

Note that a simulation box must already be defined before using the read_dump command. This can be done by the [create_box](#), [read_data](#), or [read_restart](#) commands. The read_dump command can reset the simulation box dimensions, as explained below.

Also note that reading per-atom information from a dump snapshot is limited to the atom coordinates, velocities and image flags, as explained below. Other atom properties, which may be necessary to run a valid simulation, such as atom charge, or bond topology information for a molecular system, are not read from (or even contained in) dump files. Thus this auxiliary information should be defined in the usual way, e.g. in a

data file read in by a read_data command, before using the read_dump command, or by the set command, after the dump snapshot is read.

If the dump filename specified as *file* ends with ".gz", the dump file is read in gzipped format. You cannot (yet) read a dump file that was written in binary format with a ".bin" suffix, or to multiple files via the "%" option in the dump file name. See the dump command for details.

The format of the dump file is selected through the *format* keyword. If specified, it must be the last keyword used, since all remaining arguments are passed on to the dump reader. The *native* format is for native LIGGGHTS(R)-WSU dump files, written with a "dump atom".html or dump custom command. The *xyz* format is for generic XYZ formatted dump files,

Support for other dump format readers may be added in the future.

Global information is first read from the dump file, namely timestep and box information.

The dump file is scanned for a snapshot with a time stamp that matches the specified *Nstep*. This means the LIGGGHTS(R)-WSU timestep the dump file snapshot was written on for the *native* format. However, the *xyz* formats do not store the timestep. For these formats, timesteps are numbered logically, in a sequential manner, starting from 0. Thus to access the 10th snapshot in an *xyz* or *mofile* formatted dump file, use *Nstep* = 9.

The dimensions of the simulation box for the selected snapshot are also read; see the *box* keyword discussion below. For the *native* format, an error is generated if the snapshot is for a triclinic box and the current simulation box is orthogonal or vice versa. A warning will be generated if the snapshot box boundary conditions (periodic, shrink-wrapped, etc) do not match the current simulation boundary conditions, but the boundary condition information in the snapshot is otherwise ignored. See the "boundary" command for more details.

For the *xyz* format, no information about the box is available, so you must set the *box* flag to *no*. See details below.

Per-atom information from the dump file snapshot is then read from the dump file snapshot. This corresponds to the specified *fields* listed in the read_dump command. It is an error to specify a z-dimension field, namely *z*, *vz*, or *iz*, for a 2d simulation.

For dump files in *native* format, each column of per-atom data has a text label listed in the file. A matching label for each field must appear, e.g. the label "vy" for the field *vy*. For the *x*, *y*, *z* fields any of the following labels are considered a match:

```
x, xs, xu, xsu for field x
y, ys, yu, ysu for field y
z, zs, zu, zsu for field z
```

The meaning of xs (scaled), xu (unwrapped), and xsu (scaled and unwrapped) is explained on the dump command doc page. These labels are searched for in the list of column labels in the dump file, in order, until a match is found.

The dump file must also contain atom IDs, with a column label of "id".

If the *add* keyword is specified with a value of *yes*, as discussed below, the dump file must contain atom types, with a column label of "type".

If a column label you want to read from the dump file is not a match to a specified field, the *label* keyword can be used to specify the specific column label from the dump file to associate with that field. An example is if a time-averaged coordinate is written to the dump file via the fix ave/atom command. The column will then

have a label corresponding to the fix-ID rather than "x" or "xs". The *label* keyword can also be used to specify new column labels for fields *id* and *type*.

For dump files in *xyz* format, only the *x*, *y*, and *z* fields are supported. The dump file does not store atom IDs, so these are assigned consecutively to the atoms as they appear in the dump file, starting from 1. Thus you should insure that order of atoms is consistent from snapshot to snapshot in the the XYZ dump file. See the dump_modify sort command if the XYZ dump file was written by LIGGGHTS(R)-WSU.

Information from the dump file snapshot is used to overwrite or replace properties of the current system. There are various options for how this is done, determined by the specified fields and optional keywords.

The timestep of the snapshot becomes the current timestep for the simulation. See the reset_timestep command if you wish to change this after the dump snapshot is read.

If the *box* keyword is specified with a *yes* value, then the current simulation box dimensions are replaced by the dump snapshot box dimensions. If the *box* keyword is specified with a *no* value, the current simulatoin box is unchanged.

If the *purge* keyword is specified with a *yes* value, then all current atoms in the system are deleted before any of the operations invoked by the *replace*, *trim*, or *add* keywords take place.

If the *replace* keyword is specified with a *yes* value, then atoms with IDs that are in both the current system and the dump snapshot have their properties overwritten by field values. If the *replace* keyword is specified with a *no* value, atoms with IDs that are in both the current system and the dump snapshot are not modified.

If the *trim* keyword is specified with a *yes* value, then atoms with IDs that are in the current system but not in the dump snapshot are deleted. These atoms are unaffected if the *trim* keyword is specified with a *no* value.

If the *add* keyword is specified with a *yes* value, then atoms with IDs that are in the dump snapshot, but not in the current system are added to the system. These dump atoms are ignored if the *add* keyword is specified with a *no* value.

Note that atoms added via the *add* keyword will have only the attributes read from the dump file due to the *field* arguments. If *x* or *y* or *z* is not specified as a field, a value of 0.0 is used for added atoms. Added atoms must have an atom type, so this value must appear in the dump file.

Any other attributes (e.g. charge or particle diameter for spherical particles) will be set to default values, the same as if the create_atoms command were used.

Note that atom IDs are not preserved for new dump snapshot atoms added via the *add* keyword. The procedure for assigning new atom IDS to added atoms is the same as is described for the create_atoms command.

Atom coordinates read from the dump file are first converted into unscaled coordinates, relative to the box dimensions of the snapshot. These coordinates are then be assigned to an existing or new atom in the current simulation. The coordinates will then be remapped to the simulation box, whether it is the original box or the dump snapshot box. If periodic boundary conditions apply, this means the atom will be remapped back into the simulation box if necessary. If shrink-wrap boundary conditions apply, the new coordinates may change the simulation box dimensions. If fixed boundary conditions apply, the atom will be lost if it is outside the simulation box.

For *native* format dump files, the 3 xyz image flags for an atom in the dump file are set to the corresponding values appearing in the dump file if the *ix*, *iy*, *iz* fields are specified. If not specified, the image flags for replaced atoms are not changed and image flags for new atoms are set to default values. If coordinates read

from the dump file are in unwrapped format (e.g. *xu*) then the image flags for read-in atoms are also set to default values. The remapping procedure described in the previous paragraph will then change images flags for all atoms (old and new) if periodic boundary conditions are applied to remap an atom back into the simulation box.

IMPORTANT NOTE: If you get a warning about inconsistent image flags after reading in a dump snapshot, it means one or more pairs of bonded atoms now have inconsistent image flags. As discussed in Section errors this may or may not cause problems for subsequent simulations, One way this can happen is if you read image flag fields from the dump file but do not also use the dump file box parameters.

LIGGGHTS(R)-WSU knows how to compute unscaled and remapped coordinates for the snapshot column labels discussed above, e.g. *x*, *xs*, *xu*, *xsu*. If another column label is assigned to the *x* or *y* or *z* field via the *label* keyword, e.g. for coordinates output by the fix ave/atom command, then LIGGGHTS(R)-WSU needs to know whether the coordinate information in the dump file is scaled and/or wrapped. This can be set via the *scaled* and *wrapped* keywords. Note that the value of the *scaled* and *wrapped* keywords is ignored for fields *x* or *y* or *z* if the *label* keyword is not used to assign a column label to that field.

The scaled/unscaled and wrapped/unwrapped setting must be identical for any of the *x*, *y*, *z* fields that are specified. Thus you cannot read *xs* and *yu* from the dump file. Also, if the dump file coordinates are scaled and the simulation box is triclinic, then all 3 of the *x*, *y*, *z* fields must be specified, since they are all needed to generate absolute, unscaled coordinates.

Restrictions:

To read gzipped dump files, you must compile LIGGGHTS(R)-WSU with the -DLAMMPS_GZIP option - see the Making LIGGGHTS(R)-WSU section of the documentation.

Related commands:

dump, read_data, read_restart, rerun

Default:

The option defaults are box = yes, replace = yes, purge = no, trim = no, add = no, scaled = no, wrapped = yes, and format = native.

read_restart command

Syntax:

```
read_restart file
```

- file = name of binary restart file to read in

Examples:

```
read_restart save.10000
read_restart restart.*
read_restart poly.*.%
```

Description:

Read in a previously saved simulation from a restart file. This allows continuation of a previous run. Information about what is stored in a restart file is given below.

Restart files are saved in binary format to enable exact restarts, meaning that the trajectories of a restarted run will precisely match those produced by the original run had it continued on.

Several things can prevent exact restarts due to round-off effects, in which case the trajectories in the 2 runs will slowly diverge. These include running on a different number of processors or changing certain settings such as those set by the [newton](#) or [processors](#) commands. LIGGGHTS(R)-WSU will issue a warning in these cases.

Certain fixes will not restart exactly, though they should provide statistically similar results.

Certain pair styles will not restart exactly, though they should provide statistically similar results. This is because the forces they compute depend on atom velocities, which are used at half-step values every timestep when forces are computed. When a run restarts, forces are initially evaluated with a full-step velocity, which is different than if the run had continued. These pair styles include [granular pair styles](#).

If a restarted run is immediately different than the run which produced the restart file, it could be a LIGGGHTS(R)-WSU 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. In this case, you can use the [-r command-line switch](#) to convert a restart file to a data file.

Similar to how restart files are written (see the [write_restart](#) and [restart](#) commands), the restart filename can contain two wild-card characters. If a "*" appears in the filename, the directory is searched for all filenames that match the pattern where "*" is replaced with a timestep value. The file with the largest timestep value is read in. Thus, this effectively means, read the latest restart file. It's useful if you want your script to continue a run from where it left off. See the [run](#) command and its "upto" option for how to specify the run command so it doesn't need to be changed either.

If a "%" character appears in the restart filename, LIGGGHTS(R)-WSU expects a set of multiple files to exist. The [restart](#) and [write_restart](#) commands explain how such sets are created. Read_restart will first read a filename where "%" is replaced by "base". This file tells LIGGGHTS(R)-WSU how many processors created the set and how many files are in it. Read_restart then reads the additional files. For example, if the restart file

was specified as save.% when it was written, then read_restart reads the files save.base, save.0, save.1, ... save.P-1, where P is the number of processors that created the restart file. The processors in the current LIGGGHTS(R)-WSU simulation share the work of reading these files; each reads a roughly equal subset of the files. The number of processors which created the set can be different the number of processors in the current LIGGGHTS(R)-WSU simulation. This can be a fast mode of input on parallel machines that support parallel I/O.

A restart file stores the following information about a simulation: units and atom style, simulation box size and shape and boundary settings, group definitions, per-type atom settings such as mass, per-atom attributes including their group assignments and molecular topology attributes, force field styles and coefficients, and special_bonds settings. This means that commands for these quantities do not need to be re-specified in the input script that reads the restart file, though you can redefine settings after the restart file is read.

One exception is that some pair styles do not store their info in restart files. The doc pages for individual pair styles note if this is the case. This is also true of bond_style hybrid (and angle_style, dihedral_style, improper_style hybrid).

All settings made by the pair_modify command, such as the shift and tail settings, are stored in the restart file with the pair style. The one exception is the pair_modify compute setting is not stored.

The list of fixes used for a simulation is not stored in the restart file. This means the new input script should specify all fixes it will use. Note that some fixes store an internal "state" which is written to the restart file. This allows the fix to continue on with its calculations in a restarted simulation. To re-enable such a fix, the fix command in the new input script must use the same fix-ID and group-ID as was used in the input script that wrote the restart file. If a match is found, LIGGGHTS(R)-WSU prints a message indicating that the fix is being re-enabled. If no match is found before the first run or minimization is performed by the new script, the "state" information for the saved fix is discarded. See the doc pages for individual fixes for info on which ones can be restarted in this manner.

Bonds that are broken (e.g. by a bond-breaking potential) are written to the restart file as broken bonds with a type of 0. Thus these bonds will still be broken when the restart file is read.

IMPORTANT NOTE: No other information is stored in the restart file. This means that an input script that reads a restart file should specify settings for quantities like timestep size, thermodynamic, neighbor list criteria including settings made via the neigh_modify comand, dump file output, geometric regions, etc.

Restrictions: none

Related commands:

read_data, read_dump, write_restart, restart

Default: none

region command

Syntax:

```
region ID style args keyword arg ...
```

- ID = user-assigned name for the region
- style = *delete* or *block* or *cone* or *cylinder* or *plane* or *prism* or *sphere* or *mesh/tet* or *union* or *intersect* or *wedge*

```
delete = no args
  block args = xlo xhi ylo yhi zlo zhi
    xlo,xhi,ylo,yhi,zlo,zhi = bounds of block in all dimensions (distance units)
  cone args = dim c1 c2 radlo radhi lo hi
    dim = x or y or z = axis of cone
    c1,c2 = coords of cone axis in other 2 dimensions (distance units)
    radlo,radhi = cone radii at lo and hi end (distance units)
    lo,hi = bounds of cone in dim (distance units)
  cylinder args = dim c1 c2 radius lo hi
    dim = x or y or z = axis of cylinder
    c1,c2 = coords of cylinder axis in other 2 dimensions (distance units)
    radius = cylinder radius (distance units)
      radius can be a variable (see below)
    lo,hi = bounds of cylinder in dim (distance units)
  plane args = px py pz nx ny nz
    px,py,pz = point on the plane (distance units)
    nx,ny,nz = direction normal to plane (distance units)
  prism args = xlo xhi ylo yhi zlo zhi xy xz yz
    xlo,xhi,ylo,yhi,zlo,zhi = bounds of untilted prism (distance units)
    xy = distance to tilt y in x direction (distance units)
    xz = distance to tilt z in x direction (distance units)
    yz = distance to tilt z in y direction (distance units)
  sphere args = x y z radius
    x,y,z = center of sphere (distance units)
    radius = radius of sphere (distance units)
  mesh/tet args = file filename scale s move offx offy offz rotate phix phiy phiz
    file = obligatory keyword
    filename = name of ASCII VTK file containing the VTK tet-mesh data
    scale = obligatory keyword
    s = scaling factor for the mesh (dimensionless)
    move = obligatory keyword
    offx,offy,offz = offset for the mesh (distance units)
    rotate = obligatory keyword
    phix,phiy,phiz = angle of mesh rotation around x-, y-, and z-axis (in grad)
  wedge args = axis dim center c1 c2 radius r bounds lo hi angle0 alpha0 angle alpha
    axis = obligatory keyword
    dim = x or y or z ... wedge is aligned to this dimension
    center = obligatory keyword
    c1 = first center-coordinate
          if dim == x then y-coord
          if dim == y then z-coord
          if dim == z then x-coord
    c2 = second center-coordinate
          if dim == x then z-coord
          if dim == y then x-coord
          if dim == z then y-coord
    radius = obligatory keyword
    r = radius of cylinder
    bounds = obligatory keyword
    lo = dim-coord of lower flat face of cylinder of wedge
    hi = dim-coord of higher flat face of cylinder of wedge
```

```
        angle0 = obligatory keyword
        alpha0 = mathematically positive angle of the wedge's starting face
                    if axis == x then starting from y-axis
                    if axis == y then starting from z-axis
                    if axis == z then starting from x=axis
        angle = obligatory keyword
        alpha = mathematically positive angle between the wedge's starting and ending face in
     union args = N reg-ID1 reg-ID2 ...
        N = # of regions to follow, must be 2 or greater
        reg-ID1,reg-ID2, ... = IDs of regions to join together
     intersect args = N reg-ID1 reg-ID2 ...
        N = # of regions to follow, must be 2 or greater
        reg-ID1,reg-ID2, ... = IDs of regions to intersect
```

• zero or more keyword/arg pairs may be appended

• keyword = *side* or *units* or *move* or *rotate*

```
  side value = in or out
      in = the region is inside the specified geometry
      out = the region is outside the specified geometry
    units value = lattice or box
      lattice = the geometry is defined in lattice units
      box = the geometry is defined in simulation box units
    move args = v_x v_y v_z
      v_x,v_y,v_z = equal-style variables for x,y,z displacement of region over time
    rotate args = v_theta Px Py Pz Rx Ry Rz
      v_theta = equal-style variable for rotaton of region over time (in radians)
      Px,Py,Pz = origin for axis of rotation (distance units)
      Rx,Ry,Rz = axis of rotation vector
```

Examples:

```
region 1 block -3.0 5.0 INF 10.0 INF INF
region 2 sphere 0.0 0.0 0.0 5 side out
region void cylinder y 2 3 5 -5.0 EDGE units box
region 1 prism 0 10 0 10 0 10 2 0 0
region outside union 4 side1 side2 side3 side4
region 2 sphere 0.0 0.0 0.0 5 side out move v_left v_up NULL
region 1 wedge axis y  center 0 0  radius 10 bounds 0 10 angle0 -22.5 angle 45 units box side in
```

Description:

This command defines a geometric region of space. Various other commands use regions. For example, the region can be filled with atoms via the [create_atoms](#) command. Or a bounding box around the region, can be used to define the simulation box via the [create_box](#) command. Or the atoms in the region can be identified as a group via the [group](#) command, or deleted via the [delete_atoms](#) command. Or the surface of the region can be used as a boundary wall via the [fix wall/region](#) command.

Commands which use regions typically test whether an atom's position is contained in the region or not. For this purpose, coordinates exactly on the region boundary are considered to be interior to the region. This means, for example, for a spherical region, an atom on the sphere surface would be part of the region if the sphere were defined with the *side in* keyword, but would not be part of the region if it were defined using the *side out* keyword. See more details on the *side* keyword below.

Normally, regions in LIGGGHTS(R)-WSU are "static", meaning their geometric extent does not change with time. If the *move* or *rotate* keyword is used, as described below, the region becomes "dynamic", meaning it's location or orientation changes with time. This may be useful, for example, when thermostatting a region, via the compute temp/region command, or when the fix wall/region command uses a region surface as a bounding wall on particle motion, i.e. a rotating container.

The *delete* style removes the named region. Since there is little overhead to defining extra regions, there is normally no need to do this, unless you are defining and discarding large numbers of regions in your input script.

The lo/hi values for *block* or *cone* or *cylinder* or *prism* styles can be specified as EDGE or INF. EDGE means they extend all the way to the global simulation box boundary. Note that this is the current box boundary; if the box changes size during a simulation, the region does not. INF means a large negative or positive number (1.0e20), so it should encompass the simulation box even if it changes size. If a region is defined before the simulation box has been created (via create_box or read_data or read_restart commands), then an EDGE or INF parameter cannot be used. For a *prism* region, a non-zero tilt factor in any pair of dimensions cannot be used if both the lo/hi values in either of those dimensions are INF. E.g. if the xy tilt is non-zero, then xlo and xhi cannot both be INF, nor can ylo and yhi.

IMPORTANT NOTE: Regions in LIGGGHTS(R)-WSU do not get wrapped across periodic boundaries, as specified by the boundary command. For example, a spherical region that is defined so that it overlaps a periodic boundary is not treated as 2 half-spheres, one on either side of the simulation box.

IMPORTANT NOTE: Regions in LIGGGHTS(R)-WSU are always 3d geometric objects, regardless of whether the dimension of a simulation is 2d or 3d. Thus when using regions in a 2d simulation, you should be careful to define the region so that its intersection with the 2d x-y plane of the simulation has the 2d geometric extent you want.

For style *cone*, an axis-aligned cone is defined which is like a *cylinder* except that two different radii (one at each end) can be defined. Either of the radii (but not both) can be 0.0.

For style *cone* and *cylinder*, the c1,c2 params are coordinates in the 2 other dimensions besides the cylinder axis dimension. For dim = x, c1/c2 = y/z; for dim = y, c1/c2 = x/z; for dim = z, c1/c2 = x/y. Thus the third example above specifies a cylinder with its axis in the y-direction located at x = 2.0 and z = 3.0, with a radius of 5.0, and extending in the y-direction from -5.0 to the upper box boundary.

For style *plane*, a plane is defined which contain the point (px,py,pz) and has a normal vector (nx,ny,nz). The normal vector does not have to be of unit length. The "inside" of the plane is the half-space in the direction of the normal vector; see the discussion of the *side* option below.

For style *prism*, a parallelepiped is defined (it's too hard to spell parallelepiped in an input script!). The parallelepiped has its "origin" at (xlo,ylo,zlo) and is defined by 3 edge vectors starting from the origin given by A = (xhi-xlo,0,0); B = (xy,yhi-ylo,0); C = (xz,yz,zhi-zlo). *Xy,xz,yz* can be 0.0 or positive or negative values and are called "tilt factors" because they are the amount of displacement applied to faces of an originally orthogonal box to transform it into the parallelepiped.

A prism region that will be used with the create_box command to define a triclinic simulation box must have tilt factors (xy,xz,yz) that do not skew the box more than half the distance of corresponding the parallel box length. For example, if xlo = 2 and xhi = 12, then the x box length is 10 and the xy tilt factor must be between -5 and 5. Similarly, both xz and yz must be between -(xhi-xlo)/2 and +(yhi-ylo)/2. Note that this is not a limitation, since if the maximum tilt factor is 5 (as in this example), then configurations with tilt = ..., -15, -5, 5, 15, 25, ... are all geometrically equivalent.

The *radius* value for style *sphere* and *cylinder* can be specified as an equal-style variable. If the value is a variable, it should be specified as v_name, where name is the variable name. In this case, the variable will be evaluated each timestep, and its value used to determine the radius of the region.

Equal-style variables can specify formulas with various mathematical functions, and include thermo_style command keywords for the simulation box parameters and timestep and elapsed time. Thus it is easy to specify a time-dependent radius.

region command   428

See Section_howto 12 of the doc pages for a geometric description of triclinic boxes, as defined by LIGGGHTS(R)-WSU, and how to transform these parameters to and from other commonly used triclinic representations.

For style *mesh/tet*, a tetrahedral mesh can be read from an ASCII VTK file. You can apply offset, scaling and rotation to the imported mesh via dedicated keywords. If applying more then one of these operations, the offset is applied first and then the geometry is scaled. Then the geometry is rotated around the x-axis first, then around the y-axis, then around the z-axis.

IMPORTANT NOTE: Currently only ASCII VTK containing tetrahedra are supported. For periodic boundaries, the mesh is NOT mapped. Instead, a warning is generated if a vertex lies outside the simulation box.

The *union* style creates a region consisting of the volume of all the listed regions combined. The *intersect* style creates a region consisting of the volume that is common to all the listed regions.

IMPORTANT NOTE: The *union* and *intersect* regions operate by invoking methods from their list of sub-regions. Thus you cannot delete the sub-regions after defining the *union* or *intersection* region.

The *side* keyword determines whether the region is considered to be inside or outside of the specified geometry. Using this keyword in conjunction with *union* and *intersect* regions, complex geometries can be built up. For example, if the interior of two spheres were each defined as regions, and a *union* style with *side* = out was constructed listing the region-IDs of the 2 spheres, the resulting region would be all the volume in the simulation box that was outside both of the spheres.

The *units* keyword determines the meaning of the distance units used to define the region for any argument above listed as having distance units. It also affects the scaling of the velocity vector specfied with the *vel* keyword, the amplitude vector specified with the *wiggle* keyword, and the rotation point specified with the *rotate* keyword, since they each involve a distance metric.

A *box* value selects standard distance units as defined by the units command, e.g. Angstroms for units = real or metal. A *lattice* value means the distance units are in lattice spacings. The lattice command must have been previously used to define the lattice spacings which are used as follows:

- For style *block*, the lattice spacing in dimension x is applied to xlo and xhi, similarly the spacings in dimensions y,z are applied to ylo/yhi and zlo/zhi.
- For style *cone*, the lattice spacing in argument *dim* is applied to lo and hi. The spacings in the two radial dimensions are applied to c1 and c2. The two cone radii are scaled by the lattice spacing in the dimension corresponding to c1.
- For style *cylinder*, the lattice spacing in argument *dim* is applied to lo and hi. The spacings in the two radial dimensions are applied to c1 and c2. The cylinder radius is scaled by the lattice spacing in the dimension corresponding to c1.
- For style *plane*, the lattice spacing in dimension x is applied to px and nx, similarly the spacings in dimensions y,z are applied to py/ny and pz/nz.
- For style *prism*, the lattice spacing in dimension x is applied to xlo and xhi, similarly for ylo/yhi and zlo/zhi. The lattice spacing in dimension x is applied to xy and xz, and the spacing in dimension y to yz.
- For style *sphere*, the lattice spacing in dimensions x,y,z are applied to the sphere center x,y,z. The spacing in dimension x is applied to the sphere radius.

If the *move* or *rotate* keywords are used, the region is "dynamic", meaning its location or orientation changes with time. These keywords cannot be used with a *union* or *intersect* style region. Instead, the keywords should be used to make the individual sub-regions of the *union* or *intersect* region dynamic. Normally, each sub-region should be "dynamic" in the same manner (e.g. rotate around the same point), though this is not a

region command   429

requirement.

The *move* keyword allows one or more [equal-style variables](#) to be used to specify the x,y,z displacement of the region, typically as a function of time. A variable is specified as v_name, where name is the variable name. Any of the three variables can be specified as NULL, in which case no displacement is calculated in that dimension.

Note that equal-style variables can specify formulas with various mathematical functions, and include [thermo_style](#) command keywords for the simulation box parameters and timestep and elapsed time. Thus it is easy to specify a region displacement that change as a function of time or spans consecutive runs in a continuous fashion. For the latter, see the *start* and *stop* keywords of the [run](#) command and the *elaplong* keyword of [thermo_style custom](#) for details.

For example, these commands would displace a region from its initial position, in the positive x direction, effectively at a constant velocity:

```
variable dx equal ramp(0,10)
region 2 sphere 10.0 10.0 0.0 5 move v_dx NULL NULL
```

Note that the initial displacemet is 0.0, though that is not required.

Either of these varaibles would "wiggle" the region back and forth in the y direction:

```
variable dy equal swiggle(0,5,100)
variable dysame equal 5*sin(2*PI*elaplong*dt/100)
region 2 sphere 10.0 10.0 0.0 5 move NULL v_dy NULL
```

The *rotate* keyword rotates the region around a rotation axis *R* = (Rx,Ry,Rz) that goes thru a point *P* = (Px,Py,Pz). The rotation angle is calculated, presumably as a function of time, by a variable specified as v_theta, where theta is the variable name. The variable should generate its result in radians. The direction of rotation for the region around the rotation axis is consistent with the right-hand rule: if your right-hand thumb points along *R*, then your fingers wrap around the axis in the direction of rotation.

The *move* and *rotate* keywords can be used together. In this case, the displacement specified by the *move* keyword is applied to the *P* point of the *rotate* keyword.

Restrictions:

A prism cannot be of 0.0 thickness in any dimension; use a small z thickness for 2d simulations. For 2d simulations, the xz and yz parameters must be 0.0.

Related commands:

[lattice](#), [create_atoms](#), [delete_atoms](#), [group](#)

Default:

The option defaults are side = in, units = box, and no move or rotation.

replicate command

Syntax:

```
replicate nx ny nz
```

- nx,ny,nz = replication factors in each dimension

Examples:

```
replicate 2 3 2
```

Description:

Replicate the current simulation one or more times in each dimension. For example, replication factors of 2,2,2 will create a simulation with 8x as many atoms by doubling the simulation domain in each dimension. A replication factor of 1 in a dimension leaves the simulation domain unchanged.

All properties of the atoms are replicated, including their velocities, which may or may not be desirable. New atom IDs are assigned to new atoms, as are molecule IDs. Bonds and other topology interactions are created between pairs of new atoms as well as between old and new atoms. This is done by using the image flag for each atom to "unwrap" it out of the periodic box before replicating it.

This means that any molecular bond you specify in the original data file that crosses a periodic boundary should be between two atoms with image flags that differ by 1. This will allow the bond to be unwrapped appropriately.

Restrictions:

A 2d simulation cannot be replicated in the z dimension.

If a simulation is non-periodic in a dimension, care should be used when replicating it in that dimension, as it may put atoms nearly on top of each other.

If the current simulation was read in from a restart file (before a run is performed), there can have been no fix information stored in the file for individual atoms. Similarly, no fixes can be defined at the time the replicate command is used that require vectors of atom information to be stored. This is because the replicate command does not know how to replicate that information for new atoms it creates.

Replicating a system that has rigid bodies (defined via the [fix rigid](#) command), either currently defined or that created the restart file which was read in before replicating, can cause problems if there is a bond between a pair of rigid bodies that straddle a periodic boundary. This is because the periodic image information for particles in the rigid bodies are set differently than for a non-rigid system and can result in a new bond being created that spans the periodic box. Thus you cannot use the replicate command in this scenario.

Related commands: none

Default: none

rerun command

Syntax:

```
rerun file1 file2 ... keyword args ...
```

- file1,file2,... = dump file(s) to read
- one or more keywords may be appended, keyword *dump* must appear and be last

```
keyword = first or last or every or skip or start or stop or dump
 first args = Nfirts
   Nfirst = dump timestep to start on
 last args = Nlast
   Nlast = dumptimestep to stop on
 every args = Nevery
   Nevery = read snapshots matching every this many timesteps
 skip args = Nskip
   Nskip = read one out of every Nskip snapshots
 start args = Nstart
   Nstart = timestep on which pseudo run will start
 stop args = Nstop
   Nstop = timestep to which pseudo run will end
 dump args = same as read_dump command starting with its field arguments
```

Examples:

```
rerun dump.file dump x y z vx vy vz
rerun dump1.txt dump2.txt first 10000 every 1000 dump x y z
rerun dump.vels dump x y z vx vy vz box yes format molfile lammpstrj
rerun dump.dcd dump x y z box no format molfile dcd
rerun ../run7/dump.file.gz skip 2 dump x y z box yes
```

Description:

Perform a pseudo simulation run where atom information is read one snapshot at a time from a dump file(s), and energies and forces are computed on the snapshot to produce thermodynamic or other output.

This can be useful in the following kinds of scenarios, after an initial simulation produced the dump file:

- Compute the energy and forces of snapshot using a different potential.
- Calculate one or more diagnostic quantities on the snapshots that weren't computed in the initial run. These can also be computed with settings not used in the initial run, e.g. computing an RDF via the [compute rdf](#) command with a longer cutoff than was used initially.
- Calculate the portion of per-atom forces resulting from a subset of the potential. E.g. compute only Coulombic forces. This can be done by only defining only a Coulombic pair style in the rerun script. Doing this in the original script would result in different (bad) dynamics.

Conceptually, using the rerun command is like running an input script that has a loop in it (see the [next](#) and [jump](#) commands). Each iteration of the loop reads one snapshot from the dump file via the [read_dump](#) command, sets the timestep to the appropriate value, and then invokes a [run](#) command for zero timesteps to simply compute energy and forces, and any other [thermodynamic output](#) or diagnostic info you have defined. This computation also invokes any fixes you have defined that apply constraints to the system, such as [fix shake](#) or [fix indent](#).

Note that a simulation box must already be defined before using the rerun command. This can be done by the create_box, read_data, or read_restart commands.

Also note that reading per-atom information from dump snapshots is limited to the atom coordinates, velocities and image flags as explained in the read_dump command. Other atom properties, which may be necessary to compute energies and forces, such as atom charge, or bond topology information for a molecular system, are not read from (or even contained in) dump files. Thus this auxiliary information should be defined in the usual way, e.g. in a data file read in by a read_data command, before using the rerun command.

If more than one dump file is specified, the dump files are read one after the other. It is assumed that snapshot timesteps will be in ascending order. If a snapshot is encountered that is not in ascending order, it will cause the rerun command to complete.

The *first*, *last*, *every*, *skip* keywords determine which snapshots are read from the dump file(s). Snapshots are skipped until they have a timestamp >= *Nfirst*. When a snapshot with a timestamp > *Nlast* is encountered, the rerun command finishes. Note below that the defaults for *first* and *last* are to read all snapshots. If the *every* keyword is set to a value > 0, then only snapshots with timestamps that are a multiple of *Nevery* are read (the first snapshot is always read). If *Nevery* = 0, then this criterion is ignored, i.e. every snapshot is read that meets the other criteria. If the *skip* keyword is used, then after the first snapshot is read, every Nth snapshot is read, where N = *Nskip*. E.g. if *Nskip* = 3, then only 1 out of every 3 snapshots is read, assuming the snapshot timestamp is also consistent with the other criteria.

The *start* and *stop* keywords have the same meaning that they do for the run command. They only need to be defined if (a) you are using a fix command that changes some value over time, and (b) you want the reference point for elapsed time (from start to stop) to be different than the *first* and *last* settings. See the doc page for individual fixes to see which ones can be used with the *start/stop* keywords. Note that if you define neither of the *start/stop* or *first/last* keywords, then LIGGGHTS(R)-WSU treats the pseudo run as going from 0 to a huge value (effectively infinity). This means that any quantity that a fix scales as a fraction of elapsed time in the run, will essentially remain at its intiial value.

The *dump* keyword is required and must be the last keyword specified. Its arguments are passed internally to the read_dump command. The first argument following the *dump* keyword should be the *field1* argument of the read_dump command. See the read_dump doc page for details on the various options it allows for extracting information from the dump file snapshots, and for using that information to alter the LIGGGHTS(R)-WSU simulation.

In general, a LIGGGHTS(R)-WSU input script that uses a rerun command can include and perform all the usual operations of an input script that uses the run command. There are a few exceptions and points to consider, as discussed here.

Fixes that perform time integration, such as fix nve are not invoked, since no time integration is performed.

If you only want the rerun script to perform analyses that do not involve pair interactions, such as use compute msd to calculated displacements over time, you do not need to define a pair style, which may also mean neighbor lists will not need to be calculated which saves time. The communicate cutoff command can also be used to insure ghost atoms are acquired from far enough away for operations like bond and angle evaluations, if no pair style is being used.

Every time a snapshot is read, the timestep for the simulation is reset, as if the >reset_timestep command were used. This command has some restrictions as to what fixes can be defined. See its doc page for details. For example, the fix deposit and fix dt/reset fixes are in this category. They also make no sense to use with a rerun command.

If time-averaging fixes like fix ave/time are used, they are invoked on timesteps that are a function of their *Nevery*, *Nrepeat*, and *Nfreq* settings. As an example, see the fix ave/time doc page for details. You must insure those settings are consistent with the snapshot timestamps that are read from the dump file(s). If an averaging fix is not invoked on a timestep it expects to be, LIGGGHTS(R)-WSU will flag an error.

The various forms of LIGGGHTS(R)-WSU output, as defined by the thermo_style, thermo, dump, and restart commands occur on specific timesteps. If successvive dump snapshots skip those timesteps, then no output will be produced. E.g. if you request thermodynamic output every 100 steps, but the dump file snapshots are every 1000 steps, then you will only see thermodynamic output every 1000 steps.

Restrictions:

To read gzipped dump files, you must compile LIGGGHTS(R)-WSU with the -DLAMMPS_GZIP option - see the Making LIGGGHTS(R)-WSU section of the documentation.

Related commands:

read_dump

Default:

The option defaults are first = 0, last = a huge value (effectively infinity), start = same as first, stop = same as last, every = 0, skip = 1;

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 a restart file via the [read_restart](#) command, 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:

This command cannot be used when any fixes are defined that keep track of elapsed time to perform certain kinds of time-dependent operations. All insertion commands are typically using such operations.

Various fixes use the current timestep to calculate related quantities. If the timestep is reset, this may produce unexpected behavior, but LIGGGHTS(R)-WSU allows the fixes to be defined even if the timestep is reset.

Resetting the timestep clears flags for [computes](#) that may have calculated some quantity from a previous run. This means these quantity cannot be accessed by a variable in between runs until a new run is performed. See the [variable](#) command for more details.

Related commands:

[rerun](#)

Default: none

restart command

Syntax:

```
restart 0
restart N root keyword value ...
restart N file1 file2 keyword value ...
```

- N = write a restart file every this many timesteps
- N can be a variable (see below)
- root = filename to which timestep # is appended
- file1,file2 = two full filenames, toggle between them when writing file
- zero or more keyword/value pairs may be appended
- keyword = *fileper* or *nfile*

```
  fileper arg = Np
      Np = write one file for every this many processors
    nfile arg = Nf
      Nf = write this many files, one from each of Nf processors
```

Examples:

```
restart 0
restart 1000 poly.restart
restart 1000 restart.*.equil
restart 10000 poly.%.1 poly.%.2
restart v_mystep poly.restart
```

Description:

Write out a binary restart file every so many timesteps, in either or both of two modes, as a run proceeds. A value of 0 means do not write out any restart files. The two modes are as follows. If one filename is specified, a series of filenames will be created which include the timestep in the filename. If two filenames are specified, only 2 restart files will be created, with those names. LIGGGHTS(R)-WSU will toggle between the 2 names as it writes successive restart files.

Note that you can specify the restart command twice, once with a single filename and once with two filenames. This would allow you, for example, to write out archival restart files every 100000 steps using a single filenname, and more frequent temporary restart files every 1000 steps, using two filenames. Using restart 0 will turn off both modes of output.

Similar to [dump](#) files, the restart filename(s) can contain two wild-card characters.

If a "*" appears in the single filename, it is replaced with the current timestep value. This is only recognized when a single filename is used (not when toggling back and forth). Thus, the 3rd example above creates restart files as follows: restart.1000.equil, restart.2000.equil, etc. If a single filename is used with no "*", then the timestep value is appended. E.g. the 2nd example above creates restart files as follows: poly.restart.1000, poly.restart.2000, etc.

If a "%" character appears in the restart filename(s), then one file is written for each processor and the "%" character is replaced with the processor ID from 0 to P-1. An additional file with the "%" replaced by "base" is also written, which contains global information. For example, the files written on step 1000 for filename restart.% would be restart.base.1000, restart.0.1000, restart.1.1000, ..., restart.P-1.1000. This creates smaller

files and can be a fast mode of output and subsequent input on parallel machines that support parallel I/O. The optional *fileper* and *nfile* keywords discussed below can alter the number of files written.

Restart files are written on timesteps that are a multiple of N but not on the first timestep of a run or minimization. You can use the write_restart command to write a restart file before a run begins. A restart file is not written on the last timestep of a run unless it is a multiple of N. A restart file is written on the last timestep of a minimization if N > 0 and the minimization converges.

Instead of a numeric value, N can be specifed as an equal-style variable, which should be specified as v_name, where name is the variable name. In this case, the variable is evaluated at the beginning of a run to determine the next timestep at which a restart file will be written out. On that timestep, the variable will be evaluated again to determine the next timestep, etc. Thus the variable should return timestep values. See the stagger() and logfreq() and stride() math functions for equal-style variables, as examples of useful functions to use in this context. Other similar math functions could easily be added as options for equal-style variables.

For example, the following commands will write restart files every step from 1100 to 1200, and could be useful for debugging a simulation where something goes wrong at step 1163:

```
variable        s equal stride(1100,1200,1)
restart         v_s tmp.restart
```

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, you can use the -r command-line switch to convert a restart file to a data file.

The optional *nfile* or *fileper* keywords can be used in conjunction with the "%" wildcard character in the specified restart file name(s). As explained above, the "%" character causes the restart file to be written in pieces, one piece for each of P processors. By default P = the number of processors the simulation is running on. The *nfile* or *fileper* keyword can be used to set P to a smaller value, which can be more efficient when running on a large number of processors.

The *nfile* keyword sets P to the specified Nf value. For example, if Nf = 4, and the simulation is running on 100 processors, 4 files will be written, by processors 0,25,50,75. Each will collect information from itself and the next 24 processors and write it to a restart file.

For the *fileper* keyword, the specified value of Np means write one file for every Np processors. For example, if Np = 4, every 4th processor (0,4,8,12,etc) will collect information from itself and the next 3 processors and write it to a restart file.

Restrictions: none

Related commands:

write_restart, read_restart

Default:

```
restart 0
```

run command

Syntax:

```
run N keyword values ...
```

- N = # of timesteps
- zero or more keyword/value pairs may be appended
- keyword = *upto* or *start* or *stop* or *pre* or *post* or *every*

```
  upto value = none
    start value = N1
      N1 = timestep at which 1st run started
    stop value = N2
      N2 = timestep at which last run will end
    pre value = no or yes
    post value = no or yes
    every values = M c1 c2 ...
      M = break the run into M-timestep segments and invoke one or more commands between eac
      c1,c2,...,cN = one or more LIGGGHTS(R)-WSU commands, each enclosed in quotes
      c1 = NULL means no command will be invoked
```

Examples:

```
run 10000
run 1000000 upto
run 100 start 0 stop 1000
run 1000 pre no post yes
run 100000 start 0 stop 1000000 every 1000 "print 'Protein Rg = $r'"
run 100000 every 1000 NULL
```

Description:

Run or continue dynamics for a specified number of timesteps.

When the run style is *respa*, N refers to outer loop (largest) timesteps.

A value of N = 0 is acceptable; only the thermodynamics of the system are computed and printed without taking a timestep.

The *upto* keyword means to perform a run starting at the current timestep up to the specified timestep. E.g. if the current timestep is 10,000 and "run 100000 upto" is used, then an additional 90,000 timesteps will be run. This can be useful for very long runs on a machine that allocates chunks of time and terminate your job when time is exceeded. If you need to restart your script multiple times (reading in the last restart file), you can keep restarting your script with the same run command until the simulation finally completes.

The *start* or *stop* keywords can be used if multiple runs are being performed and you want a fix command that changes some value over time to make the change across the entire set of runs and not just a single run. See the doc page for individual fixes to see which ones can be used with the *start/stop* keywords.

The *pre* and *post* keywords can be used to streamline the setup, clean-up, and associated output to the screen that happens before and after a run. This can be useful if you wish to do many short runs in succession (e.g. LIGGGHTS(R)-WSU is being called as a library which is doing other computations between successive short LIGGGHTS(R)-WSU runs).

By default (pre and post = yes), LIGGGHTS(R)-WSU creates neighbor lists, computes forces, and imposes fix constraints before every run. And after every run it gathers and prints timings statistics. If a run is just a continuation of a previous run (i.e. no settings are changed), the initial computation is not necessary; the old neighbor list is still valid as are the forces. So if *pre* is specified as "no" then the initial setup is skipped, except for printing thermodynamic info. Note that if *pre* is set to "no" for the very 1st run LIGGGHTS(R)-WSU performs, then it is overridden, since the initial setup computations must be done.

IMPORTANT NOTE: If your input script changes settings between 2 runs (e.g. adds a fix or dump or compute or changes a neighbor list parameter), then the initial setup must be performed. LIGGGHTS(R)-WSU does not check for this, but it would be an error to use the *pre no* option in this case.

If *post* is specified as "no", the full timing summary is skipped; only a one-line summary timing is printed.

The *every* keyword provides a means of breaking a LIGGGHTS(R)-WSU run into a series of shorter runs. Optionally, one or more LIGGGHTS(R)-WSU commands (c1, c2, ..., cN) will be executed in between the short runs. If used, the *every* keyword must be the last keyword, since it has a variable number of arguments. Each of the trailing arguments is a single LIGGGHTS(R)-WSU command, and each command should be enclosed in quotes, so that the entire command will be treated as a single argument. This will also prevent any variables in the command from being evaluated until it is executed multiple times during the run. Note that if a command itself needs one of its arguments quoted (e.g. the print command), then you can use a combination of single and double quotes, as in the example above or below.

The *every* keyword is a means to avoid listing a long series of runs and interleaving commands in your input script. For example, a print command could be invoked or a fix could be redefined, e.g. to reset a thermostat temperature. Or this could be useful for invoking a command you have added to LIGGGHTS(R)-WSU that wraps some other code (e.g. as a library) to perform a computation periodically during a long LIGGGHTS(R)-WSU run. See this section of the documentation for info about how to add new commands to LIGGGHTS(R)-WSU. See this section of the documentation for ideas about how to couple LIGGGHTS(R)-WSU to other codes.

With the *every* option, N total steps are simulated, in shorter runs of M steps each. After each M-length run, the specified commands are invoked. If only a single command is specified as NULL, then no command is invoked. Thus these lines:

```
variable q equal x[100]
run 6000 every 2000 "print Coord = $q"
```

are the equivalent of:

```
variable q equal x[100]
run 2000
print Coord = $q
run 2000
print Coord = $q
run 2000
print Coord = $q
```

which does 3 runs of 2000 steps and prints the x-coordinate of a particular atom between runs. Note that the variable "$q" will be evaluated afresh each time the print command is executed.

Note that by using the line continuation character "&", the run every command can be spread across many lines, though it is still a single command:

```
run 100000 every 1000 &
  "print 'Minimum value = $a'" &
  "print 'Maximum value = $b'" &
  "print 'Temp = $c'" &
```

```
"print 'Press = $d'"
```

If the *pre* and *post* options are set to "no" when used with the *every* keyword, then the 1st run will do the full setup and the last run will print the full timing summary, but these operations will be skipped for intermediate runs.

IMPORTANT NOTE: You might hope to specify a command that exits the run by jumping out of the loop, e.g.

```
variable t equal temp
run 10000 every 100 "if '$t <300.0' then 'jump SELF afterrun'"
```

Unfortunately this will not currently work. The run command simply executes each command one at a time each time it pauses, then continues the run. You can replace the jump command with a simple quit command and cause LIGGGHTS(R)-WSU to exit during the middle of a run when the condition is met.

Restrictions:

The number of specified timesteps N must fit in a signed 32-bit integer, so you are limited to slightly more than 2 billion steps (2^31) in a single run. However, you can perform successive runs to run a simulation for any number of steps (ok, up to 2^63 steps).

Related commands:

run_style,

Default:

The option defaults are start = the current timestep, stop = current timestep + N, pre = yes, and post = yes.

run_style command

Syntax:

```
run_style style args
```

- style = *verlet* or or *respa* or

```
  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
        pair or inner or middle or outer
        bond value = M
          M = which level (1-N) to compute bond 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
```

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
LIGGGHTS(R)-WSU.

The *verlet* style is a standard 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 LIGGGHTS(R)-WSU commands that specify number of timesteps (e.g. neigh_modify
parameters, dump every N timesteps, etc) refer to the outermost timesteps.

The rRESPA keywords enable you to specify at what level of the hierarchy various forces will be computed. If not specified, the defaults are that bond forces are computed at level 1 (innermost loop), angle forces are computed where bond forces are, dihedral forces are computed where angle forces are, improper forces are computed where dihedral forces are, pair forces are computed at the outermost level, and kspace forces are computed where pair forces are. The inner, middle, outer forces have no defaults.

The *inner* and *middle* keywords take additional arguments for cutoffs that are used by the pairwise force computations. If the 2 cutoffs for *inner* are 5.0 and 6.0, this means that all pairs up to 6.0 apart are computed by the inner force. Those between 5.0 and 6.0 have their force go ramped to 0.0 so the overlap with the next regime (middle or outer) is smooth. The next regime (middle or outer) will compute forces for all pairs from 5.0 outward, with those from 5.0 to 6.0 having their value ramped in an inverse manner.

Only some pair potentials support the use of the *inner* and *middle* and *outer* keywords. If not, only the *pair* keyword can be used with that pair style, meaning all pairwise forces are computed at the same rRESPA level. See the doc pages for individual pair styles for details.

When using rRESPA (or for any MD simulation) care must be taken to choose a timestep size(s) that insures the Hamiltonian for the chosen ensemble is conserved. For the constant NVE ensemble, total energy must be conserved. Unfortunately, it is difficult to know *a priori* how well energy will be conserved, and a fairly long test simulation (~10 ps) is usually necessary in order to verify that no long-term drift in energy occurs with the trial set of parameters.

With that caveat, a few rules-of-thumb may be useful in selecting *respa* settings. The following applies mostly to biomolecular simulations using the CHARMM or a similar all-atom force field, but the concepts are adaptable to other problems. Without SHAKE, bonds involving hydrogen atoms exhibit high-frequency vibrations and require a timestep on the order of 0.5 fmsec in order to conserve energy. The relatively inexpensive force computations for the bonds, angles, impropers, and dihedrals can be computed on this innermost 0.5 fmsec step. The outermost timestep cannot be greater than 4.0 fmsec without risking energy drift. Smooth switching of forces between the levels of the rRESPA hierarchy is also necessary to avoid drift, and a 1-2 angstrom "healing distance" (the distance between the outer and inner cutoffs) works reasonably well. We thus recommend the following settings for use of the *respa* style without SHAKE in biomolecular simulations:

```
timestep  4.0
run_style respa 4 2 2 2 inner 2 4.5 6.0 middle 3 8.0 10.0 outer 4
```

With these settings, users can expect good energy conservation and roughly a 2.5 fold speedup over the *verlet* style with a 0.5 fmsec timestep.

Restrictions:

Whenever using rRESPA, the user should experiment with trade-offs in speed and accuracy for their system, and verify that they are conserving energy to adequate precision.

Related commands:

timestep, run

Default:

```
run_style verlet
```

(Tuckerman) Tuckerman, Berne and Martyna, J Chem Phys, 97, p 1990 (1992).

4. Commands

This section describes how a LIGGGHTS(R)-WSU input script is formatted and the input script commands used to define a LIGGGHTS(R)-WSU simulation.

4.1 List of all commands

4.1 List of all commands

This section lists all LIGGGHTS commands alphabetically, with a separate listing below of styles within certain commands. Note that some style options for some commands are part of packages, which means they cannot be used unless the package was included when LAMMPS was built. Not all packages are included in a default build. These dependencies are listed as Restrictions in the command's documentation.

atom_modify	atom_style	bond_coeff	bond_style	boundary	box
change_box	clear	communicate	compute	compute_modify	create_atoms
create_box	delete_atoms	delete_bonds	dielectric	dimension	displace_atoms
dump	dump_modify	echo	fix	fix_modify	group
if	include	jump	label	lattice	log
mass	neigh_modify	neighbor	newton	next	orient
origin	pair_coeff	pair_style	partition	print	processors
quit	read_data	read_dump	read_restart	region	replicate
rerun	reset_timestep	restart	run	run_style	set
shell	thermo	thermo_modify	thermo_style	timestep	uncompute
undump	unfix	units	variable	velocity	write_data
write_dump	write_restart				

bond_style potentials

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

harmonic	hybrid	none

compute styles

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

atom/molecule	bond/local	centro/atom	cluster/atom
cna/atom	com	com/molecule	contact/atom
coord/atom	coord/gran	displace/atom	erotate/asphere
erotate/multisphere	erotate/sphere	erotate/sphere/atom	group/group
gyration	gyration/molecule	inertia/molecule	ke
ke/atom	ke/multisphere	msd	msd/molecule
msd/nongauss	multisphere	nparticles/tracer/region	pair/gran/local

pe	pe/atom	pressure	property/atom
property/local	property/molecule	rdf	reduce
reduce/region	rigid	slice	stress/atom
voronoi/atom	wall/gran/local		

dump styles

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

custom/vtk	image	movie

fix styles

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

adapt	addforce	ave/atom	ave/correlate
ave/euler	ave/histo	ave/spatial	ave/time
aveforce	bond/break	bond/create	box/relax
buoyancy	check/timestep/gran	deform	drag
dt/reset	efield	enforce2d	external
freeze	gravity	heat/gran	heat/gran/conductio
insert/pack	insert/rate/region	insert/stream	lineforce
massflow/mesh	mesh/surface	mesh/surface/planar	mesh/surface/stres
mesh/surface/stress/servo	momentum	move	move/mesh
multisphere	nve	nve/asphere	nve/asphere/nofor
nve/limit	nve/line	nve/noforce	nve/sphere
particledistribution/discrete	particledistribution/discrete/massbased	particledistribution/discrete/numberbased	particletemplate/multis
particletemplate/sphere	planeforce	poems	print
property/atom	property/atom/tracer	property/atom/tracer/stream	property/global
rigid	rigid/nph	rigid/npt	rigid/nve
rigid/nvt	rigid/small	setforce	sph/density/continu
sph/density/corr	sph/density/summation	sph/pressure	spring
spring/rg	spring/self	store/force	store/state
viscous	wall/gran	wall/reflect	wall/region
wall/region/sph			

pair_style potentials

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

gran	hybrid	hybrid/overlay	none
soft	sph	sph/artVisc/tensCorr	

compute styles   444

4. Commands

This section describes how a LIGGGHTS(R)-WSU input script is formatted and the input script commands used to define a LIGGGHTS(R)-WSU simulation.

4.1 List of all commands

4.1 List of all commands

everything after this line is autogenerated by autogenerate_section_commands.py

---

10. Errors

This section describes the errors you can encounter when using LIGGGHTS(R)-WSU, either conceptually, or as printed out by the program.

10.1 Common problems

If two LIGGGHTS(R)-WSU runs do not produce the same answer on different machines or different numbers of processors, this is typically not a bug. In theory you should get identical answers on any number of processors and on any machine. In practice, numerical round-off can cause slight differences and eventual divergence of molecular dynamics phase space trajectories within a few 100s or few 1000s of timesteps. However, the statistical properties of the two runs (e.g. average energy or temperature) should still be the same.

If the velocity command is used to set initial atom velocities, a particular atom can be assigned a different velocity when the problem is run on a different number of processors or on different machines. If this happens, the phase space trajectories of the two simulations will rapidly diverge. See the discussion of the *loop* option in the velocity command for details and options that avoid this issue.

Similarly, the create_atoms command generates a lattice of atoms. For the same physical system, the ordering and numbering of atoms by atom ID may be different depending on the number of processors.

Some commands use random number generators which may be setup to produce different random number streams on each processor and hence will produce different effects when run on different numbers of processors. A commonly-used example is the fix langevin command for thermostatting.

A LIGGGHTS(R)-WSU simulation typically has two stages, setup and run. Most LIGGGHTS(R)-WSU errors are detected at setup time; others like a bond stretching too far may not occur until the middle of a run.

LIGGGHTS(R)-WSU tries to flag errors and print informative error messages so you can fix the problem. Of course, LIGGGHTS(R)-WSU cannot figure out your physics or numerical mistakes, like choosing too big a timestep, specifying erroneous force field coefficients, or putting 2 atoms on top of each other! If you run into errors that LIGGGHTS(R)-WSU doesn't catch that you think it should flag, please send an email to the developers.

If you get an error message about an invalid command in your input script, you can determine what command is causing the problem by looking in the log.lammps file or using the echo command to see it on the screen. For a given command, LIGGGHTS(R)-WSU expects certain arguments in a specified order. If you mess this up, LIGGGHTS(R)-WSU 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, LIGGGHTS(R)-WSU will print a message to the screen and logfile and exit gracefully when it encounters a fatal error. Sometimes it will print a WARNING to the screen and logfile and continue on; you can decide if the WARNING is important or not. A WARNING message that is generated in the middle of a

run is only printed to the screen, not to the logfile, to avoid cluttering up thermodynamic output. If LIGGGHTS(R)-WSU 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:

LIGGGHTS(R)-WSU 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 LIGGGHTS(R)-WSU doesn't trap on those errors.

Illegal arithmetic can cause LIGGGHTS(R)-WSU 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 LIGGGHTS(R)-WSU output, something is wrong with your simulation. If you suspect this is happening, it is a good idea to print out thermodynamic info frequently (e.g. every timestep) via the thermo so you can monitor what is happening. Visualizing the atom movement is also a good idea to insure your model is behaving as you expect.

In parallel, one way LIGGGHTS(R)-WSU 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.

10.2 Reporting bugs

If you are confident that you have found a bug in LIGGGHTS(R)-WSU, follow these steps.

Check the New features and bug fixes section of the LIGGGHTS(R)-WSU WWW site to see if the bug has already been reported or fixed or the Unfixed bug to see if a fix is pending.

Check the mailing list to see if it has been discussed before.

If not, send an email to the mailing list describing the problem with any ideas you have as to what is causing it or where in the code the problem might be. The developers will ask for more info if needed, such as an input script or data files.

The most useful thing you can do to help us fix the bug is to isolate the problem. Run it on the smallest number of atoms and fewest number of processors and with the simplest input script that reproduces the bug and try to identify what command or combination of commands is causing the problem.

As a last resort, you can send an email directly to the developers.

10.3 Error & warning messages

These are two alphabetic lists of the ERROR and WARNING messages LIGGGHTS(R)-WSU prints out and the reason why. If the explanation here is not sufficient, the documentation for the offending command may help. Error and warning messages also list the source file and line number where the error was generated. For example, this message

ERROR: Illegal velocity command (velocity.cpp:78)

means that line #78 in the file src/velocity.cpp generated the error. Looking in the source code may help you figure out what went wrong.

Note that error messages from <u>user-contributed packages</u> are not listed here. If such an error occurs and is not self-explanatory, you'll need to look in the source code or contact the author of the package.

Errors:

1-3 bond count is inconsistent

        An inconsistency was detected when computing the number of 1-3 neighbors for each atom. This likely means something is wrong with the bond topologies you have defined.

1-4 bond count is inconsistent

        An inconsistency was detected when computing the number of 1-4 neighbors for each atom. This likely means something is wrong with the bond topologies you have defined.

Accelerator sharing is not currently supported on system

        Multiple MPI processes cannot share the accelerator on your system. For NVIDIA GPUs, see the nvidia-smi command to change this setting.

All angle coeffs are not set

        All angle coefficients must be set in the data file or by the angle_coeff command before running a simulation.

All bond coeffs are not set

        All bond coefficients must be set in the data file or by the bond_coeff command before running a simulation.

All dihedral coeffs are not set

        All dihedral coefficients must be set in the data file or by the dihedral_coeff command before running a simulation.

All 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 mol IDs should be set for fix gcmc group atoms

        The molecule flag is on, yet not all molecule ids in the fix group have been set to non-zero positive values by the user. This is an error since all atoms in the fix gcmc group are eligible for deletion, rotation, and translation and therefore must have valid molecule ids.

All pair coeffs are not set

        All pair coefficients must be set in the data file or by the pair_coeff command before running a simulation.

All read_dump x,y,z fields must be specified for scaled, triclinic coords

        For triclinic boxes and scaled coordinates you must specify all 3 of the x,y,z fields, else LIGGGHTS(R)-WSU cannot reconstruct the unscaled coordinates.

All universe/uloop variables must have same # of values

        Self-explanatory.

All variables in next command must be same style

        Self-explanatory.

Angle atom missing in delete_bonds

        The delete_bonds command cannot find one or more atoms in a particular angle on a particular processor. The pairwise cutoff is too short or the atoms are too far apart to make a valid angle.

Angle atom missing in set command

        The set command cannot find one or more atoms in a particular angle on a particular processor. The pairwise cutoff is too short or the atoms are too far apart to make a valid angle.

Angle atoms %d %d %d missing on proc %d at step %ld

        One or more of 3 atoms needed to compute a particular angle are missing on this processor. Typically this is because the pairwise cutoff is set too short or the angle has blown apart and an atom is too far away.

Angle coeff for hybrid has invalid style
    Angle style hybrid uses another angle style as one of its coefficients. The angle style used in the angle_coeff command or read from a restart file is not recognized.
Angle coeffs are not set
    No angle coefficients have been assigned in the data file or via the angle_coeff command.
Angle extent > half of periodic box length
    This error was detected by the neigh_modify check yes setting. It is an error because the angle atoms are so far apart it is ambiguous how it should be defined.
Angle potential must be defined for SHAKE
    When shaking angles, an angle_style potential must be used.
Angle style hybrid cannot have hybrid as an argument
    Self-explanatory.
Angle style hybrid cannot have none as an argument
    Self-explanatory.
Angle style hybrid cannot use same angle style twice
    Self-explanatory.
Angle table must range from 0 to 180 degrees
    Self-explanatory.
Angle table parameters did not set N
    List of angle table parameters must include N setting.
Angle_coeff command before angle_style is defined
    Coefficients cannot be set in the data file or via the angle_coeff command until an angle_style has been assigned.
Angle_coeff command before simulation box is defined
    The angle_coeff command cannot be used before a read_data, read_restart, or create_box command.
Angle_coeff command when no angles allowed
    The chosen atom style does not allow for angles to be defined.
Angle_style command when no angles allowed
    The chosen atom style does not allow for angles to be defined.
Angles assigned incorrectly
    Angles read in from the data file were not assigned correctly to atoms. This means there is something invalid about the topology definitions.
Angles defined but no angle types
    The data file header lists angles but no angle types.
Another input script is already being processed
    Cannot attempt to open a 2nd input script, when the original file is still being processed.
Append boundary must be shrink/minimum
    The boundary style of the face where atoms are added must be of type m (shrink/minimum).
Arccos of invalid value in variable formula
    Argument of arccos() must be between -1 and 1.
Arcsin of invalid value in variable formula
    Argument of arcsin() must be between -1 and 1.
Assigning body parameters to non-body atom
    Self-explanatory.
Assigning ellipsoid parameters to non-ellipsoid atom
    Self-explanatory.
Assigning line parameters to non-line atom
    Self-explanatory.
Assigning tri parameters to non-tri atom
    Self-explanatory.
Atom IDs must be consecutive for velocity create loop all
    Self-explanatory.
Atom count changed in fix neb
    This is not allowed in a NEB calculation.

Atom count is inconsistent, cannot write restart file

Sum of atoms across processors does not equal initial total count. This is probably because you have lost some atoms.

Atom in too many rigid bodies - boost MAXBODY

Fix poems has a parameter MAXBODY (in fix_poems.cpp) which determines the maximum number of rigid bodies a single atom can belong to (i.e. a multibody joint). The bodies you have defined exceed this limit.

Atom sort did not operate correctly

This is an internal LIGGGHTS(R)-WSU error. Please report it to the developers.

Atom sorting has bin size = 0.0

The neighbor cutoff is being used as the bin size, but it is zero. Thus you must explicitly list a bin size in the atom_modify sort command or turn off sorting.

Atom style hybrid cannot have hybrid as an argument

Self-explanatory.

Atom style hybrid cannot use same atom style twice

Self-explanatory.

Atom vector in equal-style variable formula

Atom vectors generate one value per atom which is not allowed in an equal-style variable.

Atom-style variable in equal-style variable formula

Atom-style variables generate one value per atom which is not allowed in an equal-style variable.

Atom_modify map command after simulation box is defined

The atom_modify map command cannot be used after a read_data, read_restart, or create_box command.

Atom_modify sort and first options cannot be used together

Self-explanatory.

Atom_style command after simulation box is defined

The atom_style command cannot be used after a read_data, read_restart, or create_box command.

Atom_style line can only be used in 2d simulations

Self-explanatory.

Atom_style tri can only be used in 3d simulations

Self-explanatory.

Attempt to pop empty stack in fix box/relax

Internal LIGGGHTS(R)-WSU error. Please report it to the developers.

Attempt to push beyond stack limit in fix box/relax

Internal LIGGGHTS(R)-WSU error. Please report it to the developers.

Attempting to rescale a 0.0 temperature

Cannot rescale a temperature that is already 0.0.

Bad FENE bond

Two atoms in a FENE bond have become so far apart that the bond cannot be computed.

Bad TIP4P angle type for PPPM/TIP4P

Specified angle type is not valid.

Bad TIP4P angle type for PPPMDisp/TIP4P

Specified angle type is not valid.

Bad TIP4P bond type for PPPM/TIP4P

Specified bond type is not valid.

Bad TIP4P bond type for PPPMDisp/TIP4P

Specified bond type is not valid.

Bad fix ID in fix append/atoms command

The value of the fix_id for keyword spatial must start with the suffix f_.

Bad grid of processors

The 3d grid of processors defined by the processors command does not match the number of processors LIGGGHTS(R)-WSU is being run on.

Bad kspace_modify slab parameter

Kspace_modify value for the slab/volume keyword must be >= 2.0.

Bad matrix inversion in mldivide3
> This error should not occur unless the matrix is badly formed.

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

Bad quadratic solve for particle/line collision
> This is an internal error. It should nornally not occur.

Bad quadratic solve for particle/tri collision
> This is an internal error. It should nornally not occur.

Balance command before simulation box is defined
> The balance command cannot be used before a read_data, read_restart, or create_box command.

Balance dynamic string is invalid
> The string can only contain the characters "x", "y", or "z".

Balance produced bad splits
> This should not occur. It means two or more cutting plane locations are on top of each other or out of order. Report the problem to the developers.

Bias compute does not calculate a velocity bias
> The specified compute must compute a bias for temperature.

Bias compute does not calculate temperature
> The specified compute must compute temperature.

Bias compute group does not match compute group
> The specified compute must operate on the same group as the parent compute.

Big particle in fix srd cannot be point particle
> Big particles must be extended spheriods or ellipsoids.

Bigint setting in lmptype.h is invalid
> Size of bigint is less than size of tagint.

Bigint setting in lmptype.h is not compatible
> Bigint stored in restart file is not consistent with LIGGGHTS(R)-WSU version you are running.

Bitmapped lookup tables require int/float be same size
> Cannot use pair tables on this machine, because of word sizes. Use the pair_modify command with table 0 instead.

Bitmapped table in file does not match requested table
> Setting for bitmapped table in pair_coeff command must match table in file exactly.

Bitmapped table is incorrect length in table file
> Number of table entries is not a correct power of 2.

Bond and angle potentials must be defined for TIP4P
> Cannot use TIP4P pair potential unless bond and angle potentials are defined.

Bond atom missing in box size check
> The 2nd atoms needed to compute a particular bond is 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 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 image check
> The 2nd atom in a particular bond is 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 atom missing in set command
> The set command cannot find one or more atoms in a particular bond on a particular processor. The pairwise cutoff is too short or the atoms are too far apart to make a valid bond.

Bond atoms %d %d missing on proc %d at step %ld
> The 2nd atom needed to compute a particular bond is missing on this processor. Typically this is because the pairwise cutoff is set too short or the bond has blown apart and an atom is too far away.

Bond coeff for hybrid has invalid style
> Bond style hybrid uses another bond style as one of its coefficients. The bond style used in the bond_coeff command or read from a restart file is not recognized.

Bond coeffs are not set

No bond coefficients have been assigned in the data file or via the bond_coeff command.

Bond extent > half of periodic box length

This error was detected by the neigh_modify check yes setting. It is an error because the bond atoms are so far apart it is ambiguous how it should be defined.

Bond potential must be defined for SHAKE

Cannot use fix shake unless bond potential is defined.

Bond style hybrid cannot have hybrid as an argument

Self-explanatory.

Bond style hybrid cannot have none as an argument

Self-explanatory.

Bond style hybrid cannot use same bond style twice

Self-explanatory.

Bond style quartic cannot be used with 3,4-body interactions

No angle, dihedral, or improper styles can be defined when using bond style quartic.

Bond style quartic requires special_bonds = 1,1,1

This is a restriction of the current bond quartic implementation.

Bond table parameters did not set N

List of bond table parameters must include N setting.

Bond table values are not increasing

The values in the tabulated file must be monotonically increasing.

Bond_coeff command before bond_style is defined

Coefficients cannot be set in the data file or via the bond_coeff command until an bond_style has been assigned.

Bond_coeff command before simulation box is defined

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

Bond_coeff command when no bonds allowed

The chosen atom style does not allow for bonds to be defined.

Bond_style command when no bonds allowed

The chosen atom style does not allow for bonds to be defined.

Bonds assigned incorrectly

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

Bonds defined but no bond types

The data file header lists bonds but no bond types.

Both sides of boundary must be periodic

Cannot specify a boundary as periodic only on the lo or hi side. Must be periodic on both sides.

Boundary command after simulation box is defined

The boundary command cannot be used after a read_data, read_restart, or create_box command.

Box bounds are invalid

The box boundaries specified in the read_data file are invalid. The lo value must be less than the hi value for all 3 dimensions.

Box command after simulation box is defined

The box command cannot be used after a read_data, read_restart, or create_box command.

CPU neighbor lists must be used for ellipsoid/sphere mix

When using Gay-Berne or RE-squared pair styles with both ellipsoidal and spherical particles, the neighbor list must be built on the CPU

Can not specify Pxy/Pxz/Pyz in fix box/relax with non-triclinic box

Only triclinic boxes can be used with off-diagonal pressure components. See the region prism command for details.

Can not specify Pxy/Pxz/Pyz in fix nvt/npt/nph with non-triclinic box

Only triclinic boxes can be used with off-diagonal pressure components. See the region prism command for details.

Can only use -plog with multiple partitions

Self-explanatory. See doc page discussion of command-line switches.

Can only use -pscreen with multiple partitions

Self-explanatory. See doc page discussion of command-line switches.

Can only use NEB with 1-processor replicas

This is current restriction for NEB as implemented in LIGGGHTS(R)-WSU.

Can only use TAD with 1-processor replicas for NEB

This is current restriction for NEB as implemented in LIGGGHTS(R)-WSU.

Cannot (yet) do analytic differentiation with pppm/gpu

This is a current restriction of this command.

Cannot (yet) use K-space slab correction with compute group/group

This option is not yet supported.

Cannot (yet) use Kspace slab correction with compute group/group

This option is not yet supported.

Cannot (yet) use MSM with 2d simulation

This feature is not yet supported.

Cannot (yet) use MSM with triclinic box

This feature is not yet supported.

Cannot (yet) use PPPM with triclinic box

This feature is not yet supported.

Cannot (yet) use PPPMDisp with triclinic box

This feature is not yet supported.

Cannot (yet) use single precision with MSM (remove -DFFT_SINGLE from Makefile and recompile)

Single precision cannot be used with MSM.

Cannot add atoms to fix move variable

Atoms can not be added afterwards to this fix option.

Cannot append atoms to a triclinic box

The simulation box must be defined with edges alligned with the Cartesian axes.

Cannot balance in z dimension for 2d simulation

Self-explanatory.

Cannot change box ortho/triclinic with certain fixes defined

This is because those fixes store the shape of the box. You need to use unfix to discard the fix, change the box, then redefine a new fix.

Cannot change box ortho/triclinic with dumps defined

This is because some dumps store the shape of the box. You need to use undump to discard the dump, change the box, then redefine a new dump.

Cannot change box tilt factors for orthogonal box

Cannot use tilt factors unless the simulation box is non-orthogonal.

Cannot change box to orthogonal when tilt is non-zero

Self-explanatory.

Cannot change box z boundary to nonperiodic for a 2d simulation

Self-explanatory.

Cannot change dump_modify every for dump dcd

The frequency of writing dump dcd snapshots cannot be changed.

Cannot change dump_modify every for dump xtc

The frequency of writing dump xtc snapshots cannot be changed.

Cannot change timestep once fix srd is setup

This is because various SRD properties depend on the timestep size.

Cannot change timestep with fix pour

This fix pre-computes some values based on the timestep, so it cannot be changed during a simulation run.

Cannot change_box after reading restart file with per-atom info

This is because the restart file info cannot be migrated with the atoms. You can get around this by performing a 0-timestep run which will assign the restart file info to actual atoms.

Cannot change_box in xz or yz for 2d simulation

Errors: 453

Self-explanatory.

Cannot change_box in z dimension for 2d simulation

Self-explanatory.

Cannot compute initial g_ewald_disp

LIGGGHTS(R)-WSU failed to compute an initial guess for the PPPM_disp g_ewald_6 factor that partitions the computation between real space and k-space for Disptersion interactions.

Cannot create an atom map unless atoms have IDs

The simulation requires a mapping from global atom IDs to local atoms, but the atoms that have been defined have no IDs.

Cannot create atoms with undefined lattice

Must use the lattice command before using the create_atoms command.

Cannot create/grow a vector/array of pointers for %s

LIGGGHTS(R)-WSU code is making an illegal call to the templated memory allocaters, to create a vector or array of pointers.

Cannot create_atoms after reading restart file with per-atom info

The per-atom info was stored to be used when by a fix that you may re-define. If you add atoms before re-defining the fix, then there will not be a correct amount of per-atom info.

Cannot create_box after simulation box is defined

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

Cannot currently use pair reax with pair hybrid

This is not yet supported.

Cannot currently use pppm/gpu with fix balance.

Self-explanatory.

Cannot delete group all

Self-explanatory.

Cannot delete group currently used by a compute

Self-explanatory.

Cannot delete group currently used by a dump

Self-explanatory.

Cannot delete group currently used by a fix

Self-explanatory.

Cannot delete group currently used by atom_modify first

Self-explanatory.

Cannot displace_atoms after reading restart file with per-atom info

This is because the restart file info cannot be migrated with the atoms. You can get around this by performing a 0-timestep run which will assign the restart file info to actual atoms.

Cannot do GCMC on atoms in atom_modify first group

This is a restriction due to the way atoms are organized in a list to enable the atom_modify first command.

Cannot dump JPG file

LIGGGHTS(R)-WSU was not built with the -DLAMMPS_JPEG switch in the Makefile.

Cannot dump sort on atom IDs with no atom IDs defined

Self-explanatory.

Cannot evaporate atoms in atom_modify first group

This is a restriction due to the way atoms are organized in a list to enable the atom_modify first command.

Cannot find delete_bonds group ID

Group ID used in the delete_bonds command does not exist.

Cannot have both pair_modify shift and tail set to yes

These 2 options are contradictory.

Cannot open -reorder file

Self-explanatory.

Cannot open ADP potential file %s

The specified ADP potential file cannot be opened. Check that the path and name are correct.

Cannot open AIREBO potential file %s
　The specified AIREBO potential file cannot be opened. Check that the path and name are correct.
Cannot open BOP potential file %s
　The specified BOP potential file cannot be opened. Check that the path and name are correct.
Cannot open COMB potential file %s
　The specified COMB potential file cannot be opened. Check that the path and name are correct.
Cannot open EAM potential file %s
　The specified EAM potential file cannot be opened. Check that the path and name are correct.
Cannot open EIM potential file %s
　The specified EIM potential file cannot be opened. Check that the path and name are correct.
Cannot open LCBOP potential file %s
　The specified LCBOP potential file cannot be opened. Check that the path and name are correct.
Cannot open MEAM potential file %s
　The specified MEAM potential file cannot be opened. Check that the path and name are correct.
Cannot open Stillinger-Weber potential file %s
　The specified SW potential file cannot be opened. Check that the path and name are correct.
Cannot open Tersoff potential file %s
　The specified Tersoff potential file cannot be opened. Check that the path and name are correct.
Cannot open balance output file
　Self-explanatory.
Cannot open custom file
　Self-explanatory.
Cannot open dir to search for restart file
　Using a "*" in the name of the restart file will open the current directory to search for matching file names.
Cannot open dump file
　The output file for the dump command cannot be opened. Check that the path and name are correct.
Cannot open file %s
　The specified file cannot be opened. Check that the path and name are correct.
Cannot open file variable file %s
　The specified file cannot be opened. Check that the path and name are correct.
Cannot open fix ave/correlate file %s
　The specified file cannot be opened. Check that the path and name are correct.
Cannot open fix ave/histo file %s
　The specified file cannot be opened. Check that the path and name are correct.
Cannot open fix ave/spatial file %s
　The specified file cannot be opened. Check that the path and name are correct.
Cannot open fix ave/time file %s
　The specified file cannot be opened. Check that the path and name are correct.
Cannot open fix balance output file
　Self-explanatory.
Cannot open fix poems file %s
　The specified file cannot be opened. Check that the path and name are correct.
Cannot open fix print file %s
　The output file generated by the fix print command cannot be opened
Cannot open fix qeq/comb file %s
　The output file for the fix qeq/combs command cannot be opened. Check that the path and name are correct.
Cannot open fix reax/bonds file %s
　The output file for the fix reax/bonds command cannot be opened. Check that the path and name are correct.
Cannot open fix rigid infile %s
　The specified file cannot be opened. Check that the path and name are correct.
Cannot open fix tmd file %s

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

Cannot open fix ttm file %s

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

Cannot open gzipped file

LIGGGHTS(R)-WSU is attempting to open a gzipped version of the specified file but was unsuccessful. Check that the path and name are correct.

Cannot open input script %s

Self-explanatory.

Cannot open log.lammps

The default LIGGGHTS(R)-WSU log file cannot be opened. Check that the directory you are running in allows for files to be created.

Cannot open logfile

The LIGGGHTS(R)-WSU log file named in a command-line argument cannot be opened. Check that the path and name are correct.

Cannot open logfile %s

The LIGGGHTS(R)-WSU log file specified in the input script cannot be opened. Check that the path and name are correct.

Cannot open pair_write file

The specified output file for pair energies and forces cannot be opened. Check that the path and name are correct.

Cannot open processors output file

Self-explanatory.

Cannot open restart file %s

Self-explanatory.

Cannot open screen file

The screen file specified as a command-line argument cannot be opened. Check that the directory you are running in allows for files to be created.

Cannot open universe log file

For a multi-partition run, the master log file cannot be opened. Check that the directory you are running in allows for files to be created.

Cannot open universe screen file

For a multi-partition run, the master screen file cannot be opened. Check that the directory you are running in allows for files to be created.

Cannot read_data after simulation box is defined

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

Cannot read_restart after simulation box is defined

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

Cannot redefine variable as a different style

An equal-style variable can be re-defined but only if it was originally an equal-style variable.

Cannot replicate 2d simulation in z dimension

The replicate command cannot replicate a 2d simulation in the z dimension.

Cannot replicate with fixes that store atom quantities

Either fixes are defined that create and store atom-based vectors or a restart file was read which included atom-based vectors for fixes. The replicate command cannot duplicate that information for new atoms. You should use the replicate command before fixes are applied to the system.

Cannot reset timestep with a dynamic region defined

Dynamic regions (see the region command) have a time dependence. Thus you cannot change the timestep when one or more of these are defined.

Cannot reset timestep with a time-dependent fix defined

You cannot reset the timestep when a fix that keeps track of elapsed time is in place.

Cannot 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 dump_modify flush for dump xtc
    Self-explanatory.

Cannot set mass for this atom style
    This atom style does not support mass settings for each atom type. Instead they are defined on a per-atom basis in the data file.

Cannot set meso_rho for this atom style
    Self-explanatory.

Cannot set non-zero image flag for non-periodic dimension
    Self-explanatory.

Cannot set non-zero z velocity for 2d simulation
    Self-explanatory.

Cannot set quaternion for atom that has none
    Self-explanatory.

Cannot set respa middle without inner/outer
    In the rRESPA integrator, you must define both a inner and outer setting in order to use a middle setting.

Cannot set temperature for fix rigid/nph
    The temp keyword cannot be specified.

Cannot set theta for atom that is not a line
    Self-explanatory.

Cannot set this attribute for this atom style
    The attribute being set does not exist for the defined atom style.

Cannot set variable z velocity for 2d simulation
    Self-explanatory.

Cannot skew triclinic box in z for 2d simulation
    Self-explanatory.

Cannot use -cuda on without USER-CUDA installed
    The USER-CUDA package must be installed via "make yes-user-cuda" before LIGGGHTS(R)-WSU is built.

Cannot use -reorder after -partition
    Self-explanatory. See doc page discussion of command-line switches.

Cannot use Ewald with 2d simulation
    The kspace style ewald cannot be used in 2d simulations. You can use 2d Ewald in a 3d simulation; see the kspace_modify command.

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

Cannot use Ewald/disp solver on system with no charge or LJ particles
    No atoms in system have a non-zero charge or are LJ particles. Change charges or change options of the kspace solver/pair style.

Cannot use EwaldDisp with 2d simulation
    This is a current restriction of this command.

Cannot use NEB unless atom map exists
    Use the atom_modify command to create an atom map.

Cannot use NEB with a single replica
    Self-explanatory.

Cannot use NEB with atom_modify sort enabled
    This is current restriction for NEB implemented in LIGGGHTS(R)-WSU.

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

Cannot use PPPMDisp with 2d simulation
    The kspace style pppm/disp cannot be used in 2d simulations. You can use 2d PPPM in a 3d simulation; see the kspace_modify command.

Errors: 457

Cannot use PRD with a time-dependent fix defined
> PRD alters the timestep in ways that will mess up these fixes.

Cannot use PRD with a time-dependent region defined
> PRD alters the timestep in ways that will mess up these regions.

Cannot use PRD with atom_modify sort enabled
> This is a current restriction of PRD. You must turn off sorting, which is enabled by default, via the atom_modify command.

Cannot use PRD with multi-processor replicas unless atom map exists
> Use the atom_modify command to create an atom map.

Cannot use TAD unless atom map exists for NEB
> See atom_modify map command to set this.

Cannot use TAD with a single replica for NEB
> NEB requires multiple replicas.

Cannot use TAD with atom_modify sort enabled for NEB
> This is a current restriction of NEB.

Cannot use a damped dynamics min style with fix box/relax
> This is a current restriction in LIGGGHTS(R)-WSU. Use another minimizer style.

Cannot use a damped dynamics min style with per-atom DOF
> This is a current restriction in LIGGGHTS(R)-WSU. Use another minimizer style.

Cannot use append/atoms in periodic dimension
> The boundary style of the face where atoms are added can not be of type p (periodic).

Cannot use compute cluster/atom unless atoms have IDs
> Atom IDs are used to identify clusters.

Cannot use cwiggle in variable formula between runs
> This is a function of elapsed time.

Cannot use delete_atoms unless atoms have IDs
> Your atoms do not have IDs, so the delete_atoms command cannot be used.

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

Cannot use fix GPU with USER-CUDA mode enabled
> You cannot use both the GPU and USER-CUDA packages together. Use one or the other.

Cannot use fix TMD unless atom map exists
> Using this fix requires the ability to lookup an atom index, which is provided by an atom map. An atom map does not exist (by default) for non-molecular problems. Using the atom_modify map command will force an atom map to be created.

Cannot use fix ave/spatial z for 2 dimensional model
> Self-explanatory.

Cannot use fix bond/break with non-molecular systems
> Self-explanatory.

Cannot use fix bond/create with non-molecular systems
> Self-explanatory.

Cannot use fix box/relax on a 2nd non-periodic dimension
> When specifying an off-diagonal pressure component, the 2nd of the two dimensions must be periodic. E.g. if the xy component is specified, then the y dimension must be periodic.

Cannot use fix box/relax on a non-periodic dimension
> When specifying a diagonal pressure component, the dimension must be periodic.

Cannot use fix box/relax with both relaxation and scaling on a tilt factor
> When specifying scaling on a tilt factor component, that component can not also be controlled by the barostat. E.g. if scalexy yes is specified and also keyword tri or xy, this is wrong.

Cannot use fix box/relax with tilt factor scaling on a 2nd non-periodic dimension
> When specifying scaling on a tilt factor component, the 2nd of the two dimensions must be periodic. E.g. if the xy component is specified, then the y dimension must be periodic.

Cannot use fix deform on a shrink-wrapped boundary
> The x, y, z options cannot be applied to shrink-wrapped dimensions.

Cannot use fix deform tilt on a shrink-wrapped 2nd dim
    This is because the shrink-wrapping will change the value of the strain implied by the tilt factor.
Cannot use fix deform trate on a box with zero tilt
    The trate style alters the current strain.
Cannot use fix enforce2d with 3d simulation
    Self-explanatory.
Cannot use fix gcmc in a 2d simulation
    Fix gcmc is set up to run in 3d only. No 2d simulations with fix gcmc are allowed.
Cannot use fix gcmc with a triclinic box
    Fix gcmc is set up to run with othogonal boxes only. Simulations with triclinic boxes and fix gcmc are not allowed.
Cannot use fix msst without per-type mass defined
    Self-explanatory.
Cannot use fix npt and fix deform on same component of stress tensor
    This would be changing the same box dimension twice.
Cannot use fix nvt/npt/nph on a 2nd non-periodic dimension
    When specifying an off-diagonal pressure component, the 2nd of the two dimensions must be periodic. E.g. if the xy component is specified, then the y dimension must be periodic.
Cannot use fix nvt/npt/nph on a non-periodic dimension
    When specifying a diagonal pressure component, the dimension must be periodic.
Cannot use fix nvt/npt/nph with both xy dynamics and xy scaling
    Self-explanatory.
Cannot use fix nvt/npt/nph with both xz dynamics and xz scaling
    Self-explanatory.
Cannot use fix nvt/npt/nph with both yz dynamics and yz scaling
    Self-explanatory.
Cannot use fix nvt/npt/nph with xy scaling when y is non-periodic dimension
    The 2nd dimension in the barostatted tilt factor must be periodic.
Cannot use fix nvt/npt/nph with xz scaling when z is non-periodic dimension
    The 2nd dimension in the barostatted tilt factor must be periodic.
Cannot use fix nvt/npt/nph with yz scaling when z is non-periodic dimension
    The 2nd dimension in the barostatted tilt factor must be periodic.
Cannot use fix pour with triclinic box
    This feature is not yet supported.
Cannot use fix press/berendsen and fix deform on same component of stress tensor
    These commands both change the box size/shape, so you cannot use both together.
Cannot use fix press/berendsen on a non-periodic dimension
    Self-explanatory.
Cannot use fix press/berendsen with triclinic box
    Self-explanatory.
Cannot use fix reax/bonds without pair_style reax
    Self-explantory.
Cannot use fix rigid npt/nph and fix deform on same component of stress tensor
    This would be changing the same box dimension twice.
Cannot use fix rigid npt/nph on a non-periodic dimension
    When specifying a diagonal pressure component, the dimension must be periodic.
Cannot use fix shake with non-molecular system
    Your choice of atom style does not have bonds.
Cannot use fix ttm with 2d simulation
    This is a current restriction of this fix due to the grid it creates.
Cannot use fix ttm with triclinic box
    This is a current restriction of this fix due to the grid it creates.
Cannot use fix wall in periodic dimension
    Self-explanatory.

Cannot use fix wall zlo/zhi for a 2d simulation
> Self-explanatory.

Cannot use fix wall/reflect in periodic dimension
> Self-explanatory.

Cannot use fix wall/reflect zlo/zhi for a 2d simulation
> Self-explanatory.

Cannot use fix wall/srd in periodic dimension
> Self-explanatory.

Cannot use fix wall/srd more than once
> Nor is their a need to since multiple walls can be specified in one command.

Cannot use fix wall/srd without fix srd
> Self-explanatory.

Cannot use fix wall/srd zlo/zhi for a 2d simulation
> Self-explanatory.

Cannot use force/hybrid_neigh with triclinic box
> Self-explanatory.

Cannot use force/neigh with triclinic box
> This is a current limitation of the GPU implementation in LIGGGHTS(R)-WSU.

Cannot use kspace solver on system with no charge
> No atoms in system have a non-zero charge.

Cannot use kspace solver with selected options on system with no charge
> No atoms in system have a non-zero charge. Change charges or change options of the kspace solver/pair style.

Cannot use lines with fix srd unless overlap is set
> This is because line segements are connected to each other.

Cannot use multiple fix wall commands with pair brownian
> Self-explanatory.

Cannot use multiple fix wall commands with pair lubricate
> Self-explanatory.

Cannot use multiple fix wall commands with pair lubricate/poly
> Self-explanatory.

Cannot use multiple fix wall commands with pair lubricateU
> Self-explanatory.

Cannot use neigh_modify exclude with GPU neighbor builds
> This is a current limitation of the GPU implementation in LIGGGHTS(R)-WSU.

Cannot use neighbor bins - box size << cutoff
> Too many neighbor bins will be created. This typically happens when the simulation box is very small in some dimension, compared to the neighbor cutoff. Use the "nsq" style instead of "bin" style.

Cannot use newton pair with born/coul/long/gpu pair style
> Self-explanatory.

Cannot use newton pair with born/coul/wolf/gpu pair style
> Self-explanatory.

Cannot use newton pair with born/gpu pair style
> Self-explantory.

Cannot use newton pair with buck/coul/cut/gpu pair style
> Self-explanatory.

Cannot use newton pair with buck/coul/long/gpu pair style
> Self-explanatory.

Cannot use newton pair with buck/gpu pair style
> Self-explanatory.

Cannot use newton pair with colloid/gpu pair style
> Self-explanatory.

Cannot use newton pair with coul/dsf/gpu pair style
> Self-explanatory.

Cannot use newton pair with coul/long/gpu pair style
> Self-explanatory.

Cannot use newton pair with dipole/cut/gpu pair style
> Self-explanatory.

Cannot use newton pair with eam/gpu pair style
> Self-explanatory.

Cannot use newton pair with gauss/gpu pair style
> Self-explanatory.

Cannot use newton pair with gayberne/gpu pair style
> Self-explanatory.

Cannot use newton pair with lj/charmm/coul/long/gpu pair style
> Self-explanatory.

Cannot use newton pair with lj/class2/coul/long/gpu pair style
> Self-explanatory.

Cannot use newton pair with lj/class2/gpu pair style
> Self-explanatory.

Cannot use newton pair with lj/cut/coul/cut/gpu pair style
> Self-explanatory.

Cannot use newton pair with lj/cut/coul/debye/gpu pair style
> Self-explanatory.

Cannot use newton pair with lj/cut/coul/dsf/gpu pair style
> Self-explanatory.

Cannot use newton pair with lj/cut/coul/long/gpu pair style
> Self-explanatory.

Cannot use newton pair with lj/cut/gpu pair style
> Self-explanatory.

Cannot use newton pair with lj/expand/gpu pair style
> Self-explanatory.

Cannot use newton pair with lj96/cut/gpu pair style
> Self-explanatory.

Cannot use newton pair with morse/gpu pair style
> Self-explanatory.

Cannot use newton pair with resquared/gpu pair style
> Self-explanatory.

Cannot use newton pair with table/gpu pair style
> Self-explanatory.

Cannot use newton pair with yukawa/colloid/gpu pair style
> Self-explanatory.

Cannot use newton pair with yukawa/gpu pair style
> Self-explanatory.

Cannot use non-zero forces in an energy minimization
> Fix setforce cannot be used in this manner. Use fix addforce instead.

Cannot use nonperiodic boundares with fix ttm
> This fix requires a fully periodic simulation box.

Cannot use nonperiodic boundaries with Ewald
> For kspace style ewald, all 3 dimensions must have periodic boundaries unless you use the kspace_modify command to define a 2d slab with a non-periodic z dimension.

Cannot use nonperiodic boundaries with EwaldDisp
> For kspace style ewald/disp, all 3 dimensions must have periodic boundaries unless you use the kspace_modify command to define a 2d slab with a non-periodic z dimension.

Cannot use nonperiodic boundaries with PPPM
> For kspace style pppm, all 3 dimensions must have periodic boundaries unless you use the kspace_modify command to define a 2d slab with a non-periodic z dimension.

Cannot use nonperiodic boundaries with PPPMDisp

For kspace style pppm/disp, 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 order greater than 8 with pppm/gpu.
Self-explanatory.

Cannot use pair hybrid with GPU neighbor builds
See documentation for fix gpu.

Cannot use pair tail corrections with 2d simulations
The correction factors are only currently defined for 3d systems.

Cannot use processors part command without using partitions
See the command-line -partition switch.

Cannot use ramp in variable formula between runs
This is because the ramp() function is time dependent.

Cannot use region INF or EDGE when box does not exist
Regions that extend to the box boundaries can only be used after the create_box command has been used.

Cannot use set atom with no atom IDs defined
Atom IDs are not defined, so they cannot be used to identify an atom.

Cannot use set mol with no molecule IDs defined
Self-explanatory.

Cannot use slab correction with MSM
Slab correction can only be used with Ewald and PPPM, not MSM.

Cannot use swiggle in variable formula between runs
This is a function of elapsed time.

Cannot use tris with fix srd unless overlap is set
This is because triangles are connected to each other.

Cannot use variable energy with constant force in fix addforce
This is because for constant force, LIGGGHTS(R)-WSU can compute the change in energy directly.

Cannot use variable every setting for dump dcd
The format of DCD dump files requires snapshots be output at a constant frequency.

Cannot use variable every setting for dump xtc
The format of this file requires snapshots at regular intervals.

Cannot use vdisplace in variable formula between runs
This is a function of elapsed time.

Cannot use velocity create loop all unless atoms have IDs
Atoms in the simulation to do not have IDs, so this style of velocity creation cannot be performed.

Cannot use wall in periodic dimension
Self-explanatory.

Cannot wiggle and shear fix wall/gran
Cannot specify both options at the same time.

Cannot zero Langevin force of 0 atoms
The group has zero atoms, so you cannot request its force be zeroed.

Cannot zero momentum of 0 atoms
The collection of atoms for which momentum is being computed has no atoms.

Change_box command before simulation box is defined
Self-explanatory.

Change_box volume used incorrectly
The "dim volume" option must be used immediately following one or two settings for "dim1 ..." (and optionally "dim2 ...") and must be for a different dimension, i.e. dim != dim1 and dim != dim2.

Communicate group != atom_modify first group
Self-explanatory.

Compute ID for compute atom/molecule does not exist
Self-explanatory.

Compute ID for compute reduce does not exist
Self-explanatory.

Compute ID for compute slice does not exist
    Self-explanatory.
Compute ID for fix ave/atom does not exist
    Self-explanatory.
Compute ID for fix ave/correlate does not exist
    Self-explanatory.
Compute ID for fix ave/histo does not exist
    Self-explanatory.
Compute ID for fix ave/spatial does not exist
    Self-explanatory.
Compute ID for fix ave/time does not exist
    Self-explanatory.
Compute ID for fix store/state does not exist
    Self-explanatory.
Compute ID must be alphanumeric or underscore characters
    Self-explanatory.
Compute angle/local used when angles are not allowed
    The atom style does not support angles.
Compute atom/molecule compute array is accessed out-of-range
    Self-explanatory.
Compute atom/molecule compute does not calculate a per-atom array
    Self-explanatory.
Compute atom/molecule compute does not calculate a per-atom vector
    Self-explanatory.
Compute atom/molecule compute does not calculate per-atom values
    Self-explanatory.
Compute atom/molecule fix array is accessed out-of-range
    Self-explanatory.
Compute atom/molecule fix does not calculate a per-atom array
    Self-explanatory.
Compute atom/molecule fix does not calculate a per-atom vector
    Self-explanatory.
Compute atom/molecule fix does not calculate per-atom values
    Self-explanatory.
Compute atom/molecule requires molecular atom style
    Self-explanatory.
Compute atom/molecule variable is not atom-style variable
    Self-explanatory.
Compute body/local requires atom style body
    Self-explanatory.
Compute bond/local used when bonds are not allowed
    The atom style does not support bonds.
Compute centro/atom requires a pair style be defined
    This is because the computation of the centro-symmetry values uses a pairwise neighbor list.
Compute cluster/atom cutoff is longer than pairwise cutoff
    Cannot identify clusters beyond cutoff.
Compute cluster/atom requires a pair style be defined
    This is so that the pair style defines a cutoff distance which is used to find clusters.
Compute cna/atom cutoff is longer than pairwise cutoff
    Self-explantory.
Compute cna/atom requires a pair style be defined
    Self-explantory.
Compute com/molecule requires molecular atom style
    Self-explanatory.

Compute contact/atom requires a pair style be defined
> Self-explantory.

Compute contact/atom requires atom style sphere
> Self-explanatory.

Compute coord/atom cutoff is longer than pairwise cutoff
> Cannot compute coordination at distances longer than the pair cutoff, since those atoms are not in the neighbor list.

Compute coord/atom requires a pair style be defined
> Self-explantory.

Compute damage/atom requires peridynamic potential
> Damage is a Peridynamic-specific metric. It requires you to be running a Peridynamics simulation.

Compute dihedral/local used when dihedrals are not allowed
> The atom style does not support dihedrals.

Compute does not allow an extra compute or fix to be reset
> This is an internal LIGGGHTS(R)-WSU error. Please report it to the developers.

Compute erotate/asphere requires atom style ellipsoid or line or tri
> Self-explanatory.

Compute erotate/asphere requires extended particles
> This compute cannot be used with point paritlces.

Compute erotate/sphere requires atom style sphere
> Self-explanatory.

Compute erotate/sphere/atom requires atom style sphere
> Self-explanatory.

Compute event/displace has invalid fix event assigned
> This is an internal LIGGGHTS(R)-WSU error. Please report it to the developers.

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

Compute gyration/molecule requires molecular atom style
> Self-explanatory.

Compute heat/flux compute ID does not compute ke/atom
> Self-explanatory.

Compute heat/flux compute ID does not compute pe/atom
> Self-explanatory.

Compute heat/flux compute ID does not compute stress/atom
> Self-explanatory.

Compute improper/local used when impropers are not allowed
> The atom style does not support impropers.

Compute inertia/molecule requires molecular atom style
> Self-explanatory.

Compute msd/molecule requires molecular atom style
> Self-explanatory.

Compute nve/asphere requires atom style ellipsoid
> Self-explanatory.

Compute nvt/nph/npt asphere requires atom style ellipsoid
> Self-explanatory.

Compute pair must use group all
> Pair styles accumlate energy on all atoms.

Compute pe must use group all
> Energies computed by potentials (pair, bond, etc) are computed on all atoms.

Compute pressure must use group all
> Virial contributions computed by potentials (pair, bond, etc) are computed on all atoms.

Compute pressure temperature ID does not compute temperature
> The compute ID assigned to a pressure computation must compute temperature.

Compute property/atom for atom property that isn't allocated

Self-explanatory.

Compute property/local cannot use these inputs together
> Only inputs that generate the same number of datums can be used togther. E.g. bond and angle quantities cannot be mixed.

Compute property/local for property that isn't allocated
> Self-explanatory.

Compute property/molecule requires molecular atom style
> Self-explanatory.

Compute rdf requires a pair style be defined
> Self-explanatory.

Compute reduce compute array is accessed out-of-range
> An index for the array is out of bounds.

Compute reduce compute calculates global values
> A compute that calculates peratom or local values is required.

Compute reduce compute does not calculate a local array
> Self-explanatory.

Compute reduce compute does not calculate a local vector
> Self-explanatory.

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

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

Compute reduce fix array is accessed out-of-range
> An index for the array is out of bounds.

Compute reduce fix calculates global values
> A fix that calculates peratom or local values is required.

Compute reduce fix does not calculate a local array
> Self-explanatory.

Compute reduce fix does not calculate a local vector
> Self-explanatory.

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

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

Compute reduce replace requires min or max mode
> Self-explanatory.

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

Compute slice compute array is accessed out-of-range
> An index for the array is out of bounds.

Compute slice compute does not calculate a global array
> Self-explanatory.

Compute slice compute does not calculate a global vector
> Self-explanatory.

Compute slice compute does not calculate global vector or array
> Self-explanatory.

Compute slice compute vector is accessed out-of-range
> The index for the vector is out of bounds.

Compute slice fix array is accessed out-of-range
> An index for the array is out of bounds.

Compute slice fix does not calculate a global array
> Self-explanatory.

Compute slice fix does not calculate a global vector
> Self-explanatory.

Compute slice fix does not calculate global vector or array
> Self-explanatory.

Compute slice fix vector is accessed out-of-range
> The index for the vector is out of bounds.

Compute temp/asphere requires atom style ellipsoid
> Self-explanatory.

Compute temp/asphere requires extended particles
> This compute cannot be used with point paritlces.

Compute temp/partial cannot use vz for 2d systemx
> Self-explanatory.

Compute temp/profile cannot bin z for 2d systems
> Self-explanatory.

Compute temp/profile cannot use vz for 2d systemx
> Self-explanatory.

Compute temp/sphere requires atom style sphere
> Self-explanatory.

Compute ti kspace style does not exist
> Self-explanatory.

Compute ti pair style does not exist
> Self-explanatory.

Compute ti tail when pair style does not compute tail corrections
> Self-explanatory.

Compute used in variable between runs is not current
> Computes cannot be invoked by a variable in between runs. Thus they must have been evaluated on the last timestep of the previous run in order for their value(s) to be accessed. See the doc page for the variable command for more info.

Compute used in variable thermo keyword between runs is not current
> Some thermo keywords rely on a compute to calculate their value(s). Computes cannot be invoked by a variable in between runs. Thus they must have been evaluated on the last timestep of the previous run in order for their value(s) to be accessed. See the doc page for the variable command for more info.

Compute voronoi/atom not allowed for triclinic boxes
> This is a current restriction of this command.

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

Computed temperature for fix temp/rescale cannot be 0.0
> Cannot rescale the temperature to a new value if the current temperature is 0.0.

Could not adjust g_ewald_6
> The Newton-Raphson solver failed to converge to a good value for g_ewald_6. This error should not occur for typical problems. Please send an email to the developers.

Could not compute g_ewald
> The Newton-Raphson solver failed to converge to a good value for g_ewald. This error should not occur for typical problems. Please send an email to the developers.

Could not compute grid size
> The code is unable to compute a grid size consistent with the desired accuracy. This error should not occur for typical problems. Please send an email to the developers.

Could not compute grid size for Coulomb interaction
> The code is unable to compute a grid size consistent with the desired accuracy. This error should not occur for typical problems. Please send an email to the developers.

Could not compute grid size for dispersion
> The code is unable to compute a grid size consistent with the desired accuracy. This error should not occur for typical problems. Please send an email to the developers.

Could not count initial bonds in fix bond/create
> Could not find one of the atoms in a bond on this processor.

Could not create 3d FFT plan

> The FFT setup for the PPPM solver failed, typically due to lack of memory. This is an unusual error. Check the size of the FFT grid you are requesting.

Could not create 3d grid of processors

> The specified constraints did not allow a Px by Py by Pz grid to be created where Px * Py * Pz = P = total number of processors.

Could not create 3d remap plan

> The FFT setup in pppm failed.

Could not create numa grid of processors

> The specified constraints did not allow this style of grid to be created. Usually this is because the total processor count is not a multiple of the cores/node or the user specified processor count is > 1 in one of the dimensions.

Could not create twolevel 3d grid of processors

> The specified constraints did not allow this style of grid to be created.

Could not find atom_modify first group ID

> Self-explanatory.

Could not find change_box group ID

> Group ID used in the change_box command does not exist.

Could not find compute ID for PRD

> Self-explanatory.

Could not find compute ID for TAD

> Self-explanatory.

Could not find compute ID for temperature bias

> Self-explanatory.

Could not find compute ID to delete

> Self-explanatory.

Could not find compute displace/atom fix ID

> Self-explanatory.

Could not find compute event/displace fix ID

> Self-explanatory.

Could not find compute group ID

> Self-explanatory.

Could not find compute heat/flux compute ID

> Self-explanatory.

Could not find compute msd fix ID

> Self-explanatory.

Could not find compute pressure temperature ID

> The compute ID for calculating temperature does not exist.

Could not find compute_modify ID

> Self-explanatory.

Could not find delete_atoms group ID

> Group ID used in the delete_atoms command does not exist.

Could not find delete_atoms region ID

> Region ID used in the delete_atoms command does not exist.

Could not find displace_atoms group ID

> Group ID used in the displace_atoms command does not exist.

Could not find dump custom compute ID

> The compute ID needed by dump custom to compute a per-atom quantity does not exist.

Could not find dump custom fix ID

> Self-explanatory.

Could not find dump custom variable name

> Self-explanatory.

Could not find dump group ID

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

Could not find dump local compute ID
> Self-explanatory.

Could not find dump local fix ID
> Self-explanatory.

Could not find dump modify compute ID
> Self-explanatory.

Could not find dump modify fix ID
> Self-explanatory.

Could not find dump modify variable name
> Self-explanatory.

Could not find fix ID to delete
> Self-explanatory.

Could not find fix gcmc rotation group ID
> Self-explanatory.

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

Could not find fix msst compute ID
> Self-explanatory.

Could not find fix poems group ID
> A group ID used in the fix poems command does not exist.

Could not find fix recenter group ID
> A group ID used in the fix recenter command does not exist.

Could not find fix rigid group ID
> A group ID used in the fix rigid command does not exist.

Could not find fix srd group ID
> Self-explanatory.

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

Could not find fix_modify pressure ID
> The compute ID for computing pressure does not exist.

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

Could not find group delete group ID
> Self-explanatory.

Could not find set group ID
> Group ID specified in set command does not exist.

Could not find thermo compute ID
> Compute ID specified in thermo_style command does not exist.

Could not find thermo custom compute ID
> The compute ID needed by thermo style custom to compute a requested quantity does not exist.

Could not find thermo custom fix ID
> The fix ID needed by thermo style custom to compute a requested quantity does not exist.

Could not find thermo custom variable name
> Self-explanatory.

Could not find thermo fix ID
> Fix ID specified in thermo_style command does not exist.

Could not find thermo variable name
> Self-explanatory.

Could not find thermo_modify pressure ID
> The compute ID needed by thermo style custom to compute pressure does not exist.

Could not find thermo_modify temperature ID
> The compute ID needed by thermo style custom to compute temperature does not exist.

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

Could not find velocity group ID

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

Could not find velocity temperature ID

      The compute ID needed by the velocity command to compute temperature does not exist.

Could not find/initialize a specified accelerator device

      Could not initialize at least one of the devices specified for the gpu package

Could not grab element entry from EIM potential file

      Self-explanatory

Could not grab global entry from EIM potential file

      Self-explanatory.

Could not grab pair entry from EIM potential file

      Self-explanatory.

Coulomb PPPMDisp order < minimum allowed order

      The default minimum order is 2. This can be reset by the kspace_modify minorder command.

Coulomb cut not supported in pair_style buck/long/coul/coul

      Must use long-range Coulombic interactions.

Coulomb cut not supported in pair_style lj/long/coul/long

      Must use long-range Coulombic interactions.

Coulomb cut not supported in pair_style lj/long/tip4p/long

      Must use long-range Coulombic interactions.

Coulomb cutoffs of pair hybrid sub-styles do not match

      If using a Kspace solver, all Coulomb cutoffs of long pair styles must be the same.

Cound not find dump_modify ID

      Self-explanatory.

Create_atoms command before simulation box is defined

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

Create_atoms region ID does not exist

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

Create_box region ID does not exist

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

Create_box region does not support a bounding box

      Not all regions represent bounded volumes. You cannot use such a region with the create_box command.

Cutoffs missing in pair_style buck/long/coul/long

      Self-exlanatory.

Cutoffs missing in pair_style lj/long/coul/long

      Self-explanatory.

Cyclic loop in joint connections

      Fix poems cannot (yet) work with coupled bodies whose joints connect the bodies in a ring (or cycle).

Degenerate lattice primitive vectors

      Invalid set of 3 lattice vectors for lattice command.

Delete region ID does not exist

      Self-explanatory.

Delete_atoms command before simulation box is defined

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

Delete_atoms cutoff > neighbor cutoff

      Cannot delete atoms further away than a processor knows about.

Delete_atoms requires a pair style be defined

      This is because atom deletion within a cutoff uses a pairwise neighbor list.

Delete_bonds command before simulation box is defined

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

Delete_bonds command with no atoms existing

      No atoms are yet defined so the delete_bonds command cannot be used.

Deposition region extends outside simulation box

Self-explanatory.

Did not assign all atoms correctly

Atoms read in from a data file were not assigned correctly to processors. This is likely due to some atom coordinates being outside a non-periodic simulation box.

Did not find all elements in MEAM library file

The requested elements were not found in the MEAM file.

Did not find fix shake partner info

Could not find bond partners implied by fix shake command. This error can be triggered if the delete_bonds command was used before fix shake, and it removed bonds without resetting the 1-2, 1-3, 1-4 weighting list via the special keyword.

Did not find keyword in table file

Keyword used in pair_coeff command was not found in table file.

Did not set pressure for fix rigid/nph

The press keyword must be specified.

Did not set temperature for fix rigid/nvt

The temp keyword must be specified.

Did not set temperature or pressure for fix rigid/npt

The temp and press keywords must be specified.

Dihedral atom missing in delete_bonds

The delete_bonds command cannot find one or more atoms in a particular dihedral on a particular processor. The pairwise cutoff is too short or the atoms are too far apart to make a valid dihedral.

Dihedral atom missing in set command

The set command cannot find one or more atoms in a particular dihedral on a particular processor. The pairwise cutoff is too short or the atoms are too far apart to make a valid dihedral.

Dihedral atoms %d %d %d %d missing on proc %d at step %ld

One or more of 4 atoms needed to compute a particular dihedral are missing on this processor. Typically this is because the pairwise cutoff is set too short or the dihedral has blown apart and an atom is too far away.

Dihedral charmm is incompatible with Pair style

Dihedral style charmm must be used with a pair style charmm in order for the 1-4 epsilon/sigma parameters to be defined.

Dihedral coeff for hybrid has invalid style

Dihedral style hybrid uses another dihedral style as one of its coefficients. The dihedral style used in the dihedral_coeff command or read from a restart file is not recognized.

Dihedral coeffs are not set

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

Dihedral style hybrid cannot have hybrid as an argument

Self-explanatory.

Dihedral style hybrid cannot have none as an argument

Self-explanatory.

Dihedral style hybrid cannot use same dihedral style twice

Self-explanatory.

Dihedral/improper extent > half of periodic box length

This error was detected by the neigh_modify check yes setting. It is an error because the dihedral atoms are so far apart it is ambiguous how it should be defined.

Dihedral_coeff command before dihedral_style is defined

Coefficients cannot be set in the data file or via the dihedral_coeff command until an dihedral_style has been assigned.

Dihedral_coeff command before simulation box is defined

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

Dihedral_coeff command when no dihedrals allowed

The chosen atom style does not allow for dihedrals to be defined.

Dihedral_style command when no dihedrals allowed

The chosen atom style does not allow for dihedrals to be defined.

Dihedrals assigned incorrectly

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

Dihedrals defined but no dihedral types

The data file header lists dihedrals but no dihedral types.

Dimension command after simulation box is defined

The dimension command cannot be used after a read_data, read_restart, or create_box command.

Dispersion PPPMDisp order has been reduced below minorder

This may lead to a larger grid than desired. See the kspace_modify overlap command to prevent changing of the dipsersion order.

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.

Distance must be > 0 for compute event/displace

Self-explanatory.

Divide by 0 in influence function of pair peri/lps

This should not normally occur. It is likely a problem with your model.

Divide by 0 in variable formula

Self-explanatory.

Domain too large for neighbor bins

The domain has become extremely large so that neighbor bins cannot be used. Most likely, one or more atoms have been blown out of the simulation box to a great distance.

Double precision is not supported on this accelerator

Self-explanatory

Dump cfg arguments can not mix xs|ys|zs with xsu|ysu|zsu

Self-explanatory.

Dump cfg arguments must start with 'id type xs ys zs' or 'id type xsu ysu zsu'

This is a requirement of the CFG output format.

Dump cfg requires one snapshot per file

Use the wildcard "*" character in the filename.

Dump custom and fix not computed at compatible times

The fix must produce per-atom quantities on timesteps that dump custom needs them.

Dump custom compute does not calculate per-atom array

Self-explanatory.

Dump custom compute does not calculate per-atom vector

Self-explanatory.

Dump custom compute does not compute per-atom info

Self-explanatory.

Dump custom compute vector is accessed out-of-range

Self-explanatory.

Dump custom fix does not compute per-atom array

Self-explanatory.

Dump custom fix does not compute per-atom info

Self-explanatory.

Dump custom fix does not compute per-atom vector

Self-explanatory.

Dump custom fix vector is accessed out-of-range

Self-explanatory.

Dump custom variable is not atom-style variable

Only atom-style variables generate per-atom quantities, needed for dump output.

Dump dcd of non-matching # of atoms

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

Dump dcd requires sorting by atom ID

Use the dump_modify sort command to enable this.

Dump every variable returned a bad timestep

The variable must return a timestep greater than the current timestep.

Dump file does not contain requested snapshot

Self-explanatory.

Dump file is incorrectly formatted

Self-explanatory.

Dump image bond not allowed with no bond types

Self-explanatory.

Dump image cannot perform sorting

Self-explanatory.

Dump image persp option is not yet supported

Self-explanatory.

Dump image requires one snapshot per file

Use a "*" in the filename.

Dump local and fix not computed at compatible times

The fix must produce per-atom quantities on timesteps that dump local needs them.

Dump local attributes contain no compute or fix

Self-explanatory.

Dump local cannot sort by atom ID

This is because dump local does not really dump per-atom info.

Dump local compute does not calculate local array

Self-explanatory.

Dump local compute does not calculate local vector

Self-explanatory.

Dump local compute does not compute local info

Self-explanatory.

Dump local compute vector is accessed out-of-range

Self-explanatory.

Dump local count is not consistent across input fields

Every column of output must be the same length.

Dump local fix does not compute local array

Self-explanatory.

Dump local fix does not compute local info

Self-explanatory.

Dump local fix does not compute local vector

Self-explanatory.

Dump local fix vector is accessed out-of-range

Self-explanatory.

Dump modify bcolor not allowed with no bond types

Self-explanatory.

Dump modify bdiam not allowed with no bond types

Self-explanatory.

Dump modify compute ID does not compute per-atom array

Self-explanatory.

Dump modify compute ID does not compute per-atom info

Self-explanatory.

Dump modify compute ID does not compute per-atom vector

Self-explanatory.

Dump modify compute ID vector is not large enough

Self-explanatory.

Dump modify element names do not match atom types

Number of element names must equal number of atom types.

Dump modify fix ID does not compute per-atom array

Self-explanatory.

Dump modify fix ID does not compute per-atom info

Self-explanatory.

Dump modify fix ID does not compute per-atom vector

Self-explanatory.

Dump modify fix ID vector is not large enough

Self-explanatory.

Dump modify variable is not atom-style variable

Self-explanatory.

Dump sort column is invalid

Self-explanatory.

Dump xtc requires sorting by atom ID

Use the dump_modify sort command to enable this.

Dump_modify format string is too short

There are more fields to be dumped in a line of output than your format string specifies.

Dump_modify region ID does not exist

Self-explanatory.

Dumping an atom property that isn't allocated

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

Dumping an atom quantity that isn't allocated

Only per-atom quantities that are defined for the atom style being used are allowed.

Duplicate fields in read_dump command

Self-explanatory.

Duplicate particle in PeriDynamic bond - simulation box is too small

This is likely because your box length is shorter than 2 times the bond length.

Electronic temperature dropped below zero

Something has gone wrong with the fix ttm electron temperature model.

Empty brackets in variable

There is no variable syntax that uses empty brackets. Check the variable doc page.

Energy was not tallied on needed timestep

You are using a thermo keyword that requires potentials to have tallied energy, but they didn't on this timestep. See the variable doc page for ideas on how to make this work.

Epsilon or sigma reference not set by pair style in PPPMDisp

The pair style is not providing the needed epsilon or sigma values.

Epsilon or sigma reference not set by pair style in ewald/n

The pair style is not providing the needed epsilon or sigma values.

Expected floating point parameter in input script or data file

The quantity being read is an integer on non-numeric value.

Expected floating point parameter in variable definition

The quantity being read is a non-numeric value.

Expected integer parameter in input script or data file

The quantity being read is a floating point or non-numeric value.

Expected integer parameter in variable definition

The quantity being read is a floating point or non-numeric value.

Failed to allocate %ld bytes for array %s

Your LIGGGHTS(R)-WSU simulation has run out of memory. You need to run a smaller simulation or on more processors.

Failed to reallocate %ld bytes for array %s

Your LIGGGHTS(R)-WSU simulation has run out of memory. You need to run a smaller simulation or on more processors.

Fewer SRD bins than processors in some dimension

This is not allowed. Make your SRD bin size smaller.

File variable could not read value

Check the file assigned to the variable.

Final box dimension due to fix deform is < 0.0

Self-explanatory.

Fix GPU split must be positive for hybrid pair styles

Self-explanatory.

Fix ID for compute atom/molecule does not exist

Self-explanatory.

Fix ID for compute reduce does not exist

Self-explanatory.

Fix ID for compute slice does not exist

Self-explanatory.

Fix ID for fix ave/atom does not exist

Self-explanatory.

Fix ID for fix ave/correlate does not exist

Self-explanatory.

Fix ID for fix ave/histo does not exist

Self-explanatory.

Fix ID for fix ave/spatial does not exist

Self-explanatory.

Fix ID for fix ave/time does not exist

Self-explanatory.

Fix ID for fix store/state does not exist

Self-explanatory

Fix ID for read_data does not exist

Self-explanatory.

Fix ID must be alphanumeric or underscore characters

Self-explanatory.

Fix SRD no-slip requires atom attribute torque

This is because the SRD collisions will impart torque to the solute particles.

Fix SRD: bad bin assignment for SRD advection

Something has gone wrong in your SRD model; try using more conservative settings.

Fix SRD: bad search bin assignment

Something has gone wrong in your SRD model; try using more conservative settings.

Fix SRD: bad stencil bin for big particle

Something has gone wrong in your SRD model; try using more conservative settings.

Fix SRD: too many big particles in bin

Reset the ATOMPERBIN parameter at the top of fix_srd.cpp to a larger value, and re-compile the code.

Fix SRD: too many walls in bin

This should not happen unless your system has been setup incorrectly.

Fix adapt kspace style does not exist

Self-explanatory.

Fix adapt pair style does not exist

Self-explanatory

Fix adapt pair style param not supported

The pair style does not know about the parameter you specified.

Fix adapt requires atom attribute charge

The atom style being used does not specify an atom charge.

Fix adapt requires atom attribute diameter

The atom style being used does not specify an atom diameter.

Fix adapt type pair range is not valid for pair hybrid sub-style

Self-explanatory.

Fix append/atoms requires a lattice be defined

Use the lattice command for this purpose.

Fix ave/atom compute array is accessed out-of-range

Self-explanatory.

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

Self-explanatory.

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

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

Fix ave/atom compute does not calculate per-atom values

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

Fix ave/atom fix array is accessed out-of-range

Self-explanatory.

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

Self-explanatory.

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

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

Fix ave/atom fix does not calculate per-atom values

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

Fix ave/atom missed timestep

You cannot reset the timestep to a value beyond where the fix expects to next perform averaging.

Fix ave/atom variable is not atom-style variable

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

Fix ave/correlate compute does not calculate a scalar

Self-explanatory.

Fix ave/correlate compute does not calculate a vector

Self-explanatory.

Fix ave/correlate compute vector is accessed out-of-range

The index for the vector is out of bounds.

Fix ave/correlate fix does not calculate a scalar

Self-explanatory.

Fix ave/correlate fix does not calculate a vector

Self-explanatory.

Fix ave/correlate fix vector is accessed out-of-range

The index for the vector is out of bounds.

Fix ave/correlate missed timestep

You cannot reset the timestep to a value beyond where the fix expects to next perform averaging.

Fix ave/correlate variable is not equal-style variable

Self-explanatory.

Fix ave/histo cannot input local values in scalar mode

Self-explanatory.

Fix ave/histo cannot input per-atom values in scalar mode

Self-explanatory.

Fix ave/histo compute array is accessed out-of-range

Self-explanatory.

Fix ave/histo compute does not calculate a global array

Self-explanatory.

Fix ave/histo compute does not calculate a global scalar

Self-explanatory.

Fix ave/histo compute does not calculate a global vector

Self-explanatory.

Fix ave/histo compute does not calculate a local array

Self-explanatory.

Fix ave/histo compute does not calculate a local vector

Self-explanatory.

Fix ave/histo compute does not calculate a per-atom array

Self-explanatory.

Fix ave/histo compute does not calculate a per-atom vector

Self-explanatory.

Fix ave/histo compute does not calculate local values

Self-explanatory.

Fix ave/histo compute does not calculate per-atom values

Self-explanatory.

Fix ave/histo compute vector is accessed out-of-range

Self-explanatory.

Fix ave/histo fix array is accessed out-of-range

Self-explanatory.

Fix ave/histo fix does not calculate a global array

Self-explanatory.

Fix ave/histo fix does not calculate a global scalar

Self-explanatory.

Fix ave/histo fix does not calculate a global vector

Self-explanatory.

Fix ave/histo fix does not calculate a local array

Self-explanatory.

Fix ave/histo fix does not calculate a local vector

Self-explanatory.

Fix ave/histo fix does not calculate a per-atom array

Self-explanatory.

Fix ave/histo fix does not calculate a per-atom vector

Self-explanatory.

Fix ave/histo fix does not calculate local values

Self-explanatory.

Fix ave/histo fix does not calculate per-atom values

Self-explanatory.

Fix ave/histo fix vector is accessed out-of-range

Self-explanatory.

Fix ave/histo input is invalid compute

Self-explanatory.

Fix ave/histo input is invalid fix

Self-explanatory.

Fix ave/histo input is invalid variable

Self-explanatory.

Fix ave/histo inputs are not all global, peratom, or local

All inputs in a single fix ave/histo command must be of the same style.

Fix ave/histo missed timestep

You cannot reset the timestep to a value beyond where the fix expects to next perform averaging.

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

Self-explanatory.

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

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

Fix ave/spatial compute does not calculate per-atom values

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

Fix ave/spatial compute vector is accessed out-of-range

The index for the vector is out of bounds.

Fix ave/spatial fix does not calculate a per-atom array

Self-explanatory.

Fix ave/spatial fix does not calculate a per-atom vector

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

Fix ave/spatial fix does not calculate per-atom values

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

Fix ave/spatial fix vector is accessed out-of-range

The index for the vector is out of bounds.

Fix ave/spatial for triclinic boxes requires units reduced
> Self-explanatory.

Fix ave/spatial missed timestep
> You cannot reset the timestep to a value beyond where the fix expects to next perform averaging.

Fix ave/spatial settings invalid with changing box
> If the ave setting is "running" or "window" and the box size/shape changes during the simulation, then the units setting must be "reduced", else the number of bins may change.

Fix ave/spatial variable is not atom-style variable
> A variable used by fix ave/spatial must generate per-atom values.

Fix ave/time cannot set output array intensive/extensive from these inputs
> One of more of the vector inputs has individual elements which are flagged as intensive or extensive. Such an input cannot be flagged as all intensive/extensive when turned into an array by fix ave/time.

Fix ave/time cannot use variable with vector mode
> Variables produce scalar values.

Fix ave/time columns are inconsistent lengths
> Self-explanatory.

Fix ave/time compute array is accessed out-of-range
> An index for the array is out of bounds.

Fix ave/time compute does not calculate a scalar
> Self-explantory.

Fix ave/time compute does not calculate a vector
> Self-explantory.

Fix ave/time compute does not calculate an array
> Self-explanatory.

Fix ave/time compute vector is accessed out-of-range
> The index for the vector is out of bounds.

Fix ave/time fix array is accessed out-of-range
> An index for the array is out of bounds.

Fix ave/time fix does not calculate a scalar
> Self-explanatory.

Fix ave/time fix does not calculate a vector
> Self-explanatory.

Fix ave/time fix does not calculate an array
> Self-explanatory.

Fix ave/time fix vector is accessed out-of-range
> The index for the vector is out of bounds.

Fix ave/time missed timestep
> You cannot reset the timestep to a value beyond where the fix expects to next perform averaging.

Fix ave/time variable is not equal-style variable
> Self-explanatory.

Fix balance string is invalid
> The string can only contain the characters "x", "y", or "z".

Fix balance string is invalid for 2d simulation
> The string cannot contain the letter "z".

Fix bond/break requires special_bonds = 0,1,1
> This is a restriction of the current fix bond/break implementation.

Fix bond/create cutoff is longer than pairwise cutoff
> This is not allowed because bond creation is done using the pairwise neighbor list.

Fix bond/create requires special_bonds coul = 0,1,1
> Self-explanatory.

Fix bond/create requires special_bonds lj = 0,1,1
> Self-explanatory.

Fix bond/swap cannot use dihedral or improper styles

These styles cannot be defined when using this fix.

Fix bond/swap requires pair and bond styles

Self-explanatory.

Fix bond/swap requires special_bonds = 0,1,1

Self-explanatory.

Fix box/relax generated negative box length

The pressure being applied is likely too large. Try applying it incrementally, to build to the high pressure.

Fix command before simulation box is defined

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

Fix deform cannot use yz variable with xy

The yz setting cannot be a variable if xy deformation is also specified. This is because LIGGGHTS(R)-WSU cannot determine if the yz setting will induce a box flip which would be invalid if xy is also changing.

Fix deform is changing yz too much with xy

When both yz and xy are changing, it induces changes in xz if the box must flip from one tilt extreme to another. Thus it is not allowed for yz to grow so much that a flip is induced.

Fix deform tilt factors require triclinic box

Cannot deform the tilt factors of a simulation box unless it is a triclinic (non-orthogonal) box.

Fix deform volume setting is invalid

Cannot use volume style unless other dimensions are being controlled.

Fix deposit region cannot be dynamic

Only static regions can be used with fix deposit.

Fix deposit region does not support a bounding box

Not all regions represent bounded volumes. You cannot use such a region with the fix deposit command.

Fix efield requires atom attribute q

Self-explanatory.

Fix evaporate molecule requires atom attribute molecule

The atom style being used does not define a molecule ID.

Fix external callback function not set

This must be done by an external program in order to use this fix.

Fix for fix ave/atom not computed at compatible time

Fixes generate their values on specific timesteps. Fix ave/atom is requesting a value on a non-allowed timestep.

Fix for fix ave/correlate not computed at compatible time

Fixes generate their values on specific timesteps. Fix ave/correlate is requesting a value on a non-allowed timestep.

Fix for fix ave/histo not computed at compatible time

Fixes generate their values on specific timesteps. Fix ave/histo is requesting a value on a non-allowed timestep.

Fix for fix ave/spatial not computed at compatible time

Fixes generate their values on specific timesteps. Fix ave/spatial is requesting a value on a non-allowed timestep.

Fix for fix ave/time not computed at compatible time

Fixes generate their values on specific timesteps. Fix ave/time is requesting a value on a non-allowed timestep.

Fix for fix store/state not computed at compatible time

Fixes generate their values on specific timesteps. Fix store/state is requesting a value on a non-allowed timestep.

Fix freeze requires atom attribute torque

The atom style defined does not have this attribute.

Fix gcmc cannot exchange individual atoms belonging to a molecule

This is an error since you should not delete only one atom of a molecule. The user has specified

atomic (non-molecular) gas exchanges, but an atom belonging to a molecule could be deleted.

Fix gcmc could not find any atoms in the user-supplied template molecule

When using the molecule option with fix gcmc, the user must supply a template molecule in the usual LIGGGHTS(R)-WSU data file with its molecule id specified in the fix gcmc command as the "type" of the exchanged gas.

Fix gcmc incompatible with given pair_style

Some pair_styles do not provide single-atom energies, which are needed by fix gcmc.

Fix gcmc incorrect number of atoms per molecule

The number of atoms in each gas molecule was not computed correctly.

Fix gcmc molecule command requires that atoms have molecule attributes

Should not choose the GCMC molecule feature if no molecules are being simulated. The general molecule flag is off, but GCMC's molecule flag is on.

Fix gcmc ran out of available molecule IDs

This is a code limitation where more than MAXSMALLINT (usually around two billion) molecules have been created. The code needs to be modified to either allow molecule ID recycling or use bigger ints for molecule IDs. A work-around is to run shorter simulations.

Fix gcmc region cannot be dynamic

Only static regions can be used with fix gcmc.

Fix gcmc region does not support a bounding box

Not all regions represent bounded volumes. You cannot use such a region with the fix gcmc command.

Fix gcmc region extends outside simulation box

Self-explanatory.

Fix heat group has no atoms

Self-explanatory.

Fix heat kinetic energy of an atom went negative

This will cause the velocity rescaling about to be performed by fix heat to be invalid.

Fix heat kinetic energy went negative

This will cause the velocity rescaling about to be performed by fix heat to be invalid.

Fix in variable not computed at compatible time

Fixes generate their values on specific timesteps. The variable is requesting the values on a non-allowed timestep.

Fix langevin angmom requires atom style ellipsoid

Self-explanatory.

Fix langevin angmom requires extended particles

This fix option cannot be used with point paritlces.

Fix langevin omega requires atom style sphere

Self-explanatory.

Fix langevin omega requires extended particles

One of the particles has radius 0.0.

Fix langevin period must be > 0.0

The time window for temperature relaxation must be > 0

Fix langevin variable returned negative temperature

Self-explanatory.

Fix momentum group has no atoms

Self-explanatory.

Fix move cannot define z or vz variable for 2d problem

Self-explanatory.

Fix move cannot rotate aroung non z-axis for 2d problem

Self-explanatory.

Fix move cannot set linear z motion for 2d problem

Self-explanatory.

Fix move cannot set wiggle z motion for 2d problem

Self-explanatory.

Fix msst compute ID does not compute potential energy
> Self-explanatory.

Fix msst compute ID does not compute pressure
> Self-explanatory.

Fix msst compute ID does not compute temperature
> Self-explanatory.

Fix msst requires a periodic box
> Self-explanatory.

Fix msst tscale must satisfy 0 <= tscale < 1
> Self-explanatory.

Fix npt/nph has tilted box too far in one step - periodic cell is too far from equilibrium state
> Self-explanatory. The change in the box tilt is too extreme on a short timescale.

Fix nve/asphere requires extended particles
> This fix can only be used for particles with a shape setting.

Fix nve/asphere/noforce requires atom style ellipsoid
> Self-explanatory.

Fix nve/asphere/noforce requires extended particles
> One of the particles is not an ellipsoid.

Fix nve/body requires atom style body
> Self-explanatory.

Fix nve/body requires bodies
> This fix can only be used for particles that are bodies.

Fix nve/line can only be used for 2d simulations
> Self-explanatory.

Fix nve/line requires atom style line
> Self-explanatory.

Fix nve/line requires line particles
> Self-explanatory.

Fix nve/sphere requires atom attribute mu
> An atom style with this attribute is needed.

Fix nve/sphere requires atom style sphere
> Self-explanatory.

Fix nve/sphere requires extended particles
> This fix can only be used for particles of a finite size.

Fix nve/tri can only be used for 3d simulations
> Self-explanatory.

Fix nve/tri requires atom style tri
> Self-explanatory.

Fix nve/tri requires tri particles
> Self-explanatory.

Fix nvt/nph/npt asphere requires extended particles
> The shape setting for a particle in the fix group has shape = 0.0, which means it is a point particle.

Fix nvt/nph/npt sphere requires atom style sphere
> Self-explanatory.

Fix nvt/npt/nph damping parameters must be > 0.0
> Self-explanatory.

Fix nvt/npt/nph dilate group ID does not exist
> Self-explanatory.

Fix nvt/sphere requires extended particles
> This fix can only be used for particles of a finite size.

Fix orient/fcc file open failed
> The fix orient/fcc command could not open a specified file.

Fix orient/fcc file read failed
> The fix orient/fcc command could not read the needed parameters from a specified file.

Fix orient/fcc found self twice
>   The neighbor lists used by fix orient/fcc are messed up. If this error occurs, it is likely a bug, so send an email to the [developers](#).

Fix peri neigh does not exist
>   Somehow a fix that the pair style defines has been deleted.

Fix pour region ID does not exist
>   Self-explanatory.

Fix pour region cannot be dynamic
>   Only static regions can be used with fix pour.

Fix pour region does not support a bounding box
>   Not all regions represent bounded volumes. You cannot use such a region with the fix pour command.

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

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

Fix qeq/comb group has no atoms
>   Self-explanatory.

Fix qeq/comb requires atom attribute q
>   An atom style with charge must be used to perform charge equilibration.

Fix reax/bonds numbonds > nsbmax_most
>   The limit of the number of bonds expected by the ReaxFF force field was exceeded.

Fix recenter group has no atoms
>   Self-explanatory.

Fix restrain requires an atom map, see atom_modify
>   Self-explanatory.

Fix rigid atom has non-zero image flag in a non-periodic dimension
>   Image flags for non-periodic dimensions should not be set.

Fix rigid langevin period must be > 0.0
>   Self-explanatory.

Fix rigid molecule requires atom attribute molecule
>   Self-explanatory.

Fix rigid npt/nph dilate group ID does not exist
>   Self-explanatory.

Fix rigid npt/nph does not yet allow triclinic box
>   Self-explanatory.

Fix rigid npt/nph period must be > 0.0
>   Self-explanatory.

Fix rigid nvt/npt/nph damping parameters must be > 0.0
>   Self-explanatory.

Fix rigid xy torque cannot be on for 2d simulation
>   Self-explanatory.

Fix rigid z force cannot be on for 2d simulation
>   Self-explanatory.

Fix rigid/npt period must be > 0.0
>   Self-explanatory.

Fix rigid/npt temperature order must be 3 or 5
>   Self-explanatory.

Fix rigid/nvt period must be > 0.0
>   Self-explanatory.

Fix rigid/nvt temperature order must be 3 or 5
>   Self-explanatory.

Fix rigid/small atom has non-zero image flag in a non-periodic dimension
>   Image flags for non-periodic dimensions should not be set.

Fix rigid/small langevin period must be > 0.0

Self-explanatory.

Fix rigid/small requires atom attribute molecule

Self-explanatory.

Fix rigid: Bad principal moments

The principal moments of inertia computed for a rigid body are not within the required tolerances.

Fix shake cannot be used with minimization

Cannot use fix shake while doing an energy minimization since it turns off bonds that should contribute to the energy.

Fix spring couple group ID does not exist

Self-explanatory.

Fix srd lamda must be >= 0.6 of SRD grid size

This is a requirement for accuracy reasons.

Fix srd requires SRD particles all have same mass

Self-explanatory.

Fix srd requires ghost atoms store velocity

Use the communicate vel yes command to enable this.

Fix srd requires newton pair on

Self-explanatory.

Fix store/state compute array is accessed out-of-range

Self-explanatory.

Fix store/state compute does not calculate a per-atom array

The compute calculates a per-atom vector.

Fix store/state compute does not calculate a per-atom vector

The compute calculates a per-atom vector.

Fix store/state compute does not calculate per-atom values

Computes that calculate global or local quantities cannot be used with fix store/state.

Fix store/state fix array is accessed out-of-range

Self-explanatory.

Fix store/state fix does not calculate a per-atom array

The fix calculates a per-atom vector.

Fix store/state fix does not calculate a per-atom vector

The fix calculates a per-atom array.

Fix store/state fix does not calculate per-atom values

Fixes that calculate global or local quantities cannot be used with fix store/state.

Fix store/state for atom property that isn't allocated

Self-explanatory.

Fix store/state variable is not atom-style variable

Only atom-style variables calculate per-atom quantities.

Fix temp/berendsen period must be > 0.0

Self-explanatory.

Fix temp/berendsen variable returned negative temperature

Self-explanatory.

Fix temp/rescale variable returned negative temperature

Self-explanatory.

Fix thermal/conductivity swap value must be positive

Self-explanatory.

Fix tmd must come after integration fixes

Any fix tmd command must appear in the input script after all time integration fixes (nve, nvt, npt). See the fix tmd documentation for details.

Fix ttm electron temperatures must be > 0.0

Self-explanatory.

Fix ttm electronic_density must be > 0.0

Self-explanatory.

Fix ttm electronic_specific_heat must be > 0.0

Self-explanatory.

Fix ttm electronic_thermal_conductivity must be >= 0.0

Self-explanatory.

Fix ttm gamma_p must be > 0.0

Self-explanatory.

Fix ttm gamma_s must be >= 0.0

Self-explanatory.

Fix ttm number of nodes must be > 0

Self-explanatory.

Fix ttm v_0 must be >= 0.0

Self-explanatory.

Fix used in compute atom/molecule not computed at compatible time

The fix must produce per-atom quantities on timesteps that the compute needs them.

Fix used in compute reduce not computed at compatible time

Fixes generate their values on specific timesteps. Compute reduce is requesting a value on a non-allowed timestep.

Fix used in compute slice not computed at compatible time

Fixes generate their values on specific timesteps. Compute slice is requesting a value on a non-allowed timestep.

Fix viscosity swap value must be positive

Self-explanatory.

Fix viscosity vtarget value must be positive

Self-explanatory.

Fix wall cutoff <= 0.0

Self-explanatory.

Fix wall/colloid requires atom style sphere

Self-explanatory.

Fix wall/colloid requires extended particles

One of the particles has radius 0.0.

Fix wall/gran is incompatible with Pair style

Must use a granular pair style to define the parameters needed for this fix.

Fix wall/gran requires atom style sphere

Self-explanatory.

Fix wall/piston command only available at zlo

The face keyword must be zlo.

Fix wall/region colloid requires atom style sphere

Self-explanatory.

Fix wall/region colloid requires extended particles

One of the particles has radius 0.0.

Fix wall/region cutoff <= 0.0

Self-explanatory.

Fix_modify pressure ID does not compute pressure

The compute ID assigned to the fix must compute pressure.

Fix_modify temperature ID does not compute temperature

The compute ID assigned to the fix must compute temperature.

For triclinic deformation, specified target stress must be hydrostatic

Triclinic pressure control is allowed using the tri keyword, but non-hydrostatic pressure control can not be used in this case.

Found no restart file matching pattern

When using a "*" in the restart file name, no matching file was found.

GPU library not compiled for this accelerator

Self-explanatory.

GPU particle split must be set to 1 for this pair style.

For this pair style, you cannot run part of the force calculation on the host. See the package command.

Gmask function in equal-style variable formula

  Gmask is per-atom operation.

Gravity changed since fix pour was created

  Gravity must be static and not dynamic for use with fix pour.

Gravity must point in -y to use with fix pour in 2d

  Gravity must be pointing "down" in a 2d box.

Gravity must point in -z to use with fix pour in 3d

  Gravity must be pointing "down" in a 3d box, i.e. theta = 180.0.

Grmask function in equal-style variable formula

  Grmask is per-atom operation.

Group ID does not exist

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

Group ID in variable formula does not exist

  Self-explanatory.

Group command before simulation box is defined

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

Group region ID does not exist

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

If read_dump purges it cannot replace or trim

  These operations are not compatible. See the read_dump doc page for details.

Illegal ... command

  Self-explanatory. Check the input script syntax and compare to the documentation for the command. You can use -echo screen as a command-line option when running LIGGGHTS(R)-WSU to see the offending line.

Illegal COMB parameter

  One or more of the coefficients defined in the potential file is invalid.

Illegal Stillinger-Weber parameter

  One or more of the coefficients defined in the potential file is invalid.

Illegal Tersoff parameter

  One or more of the coefficients defined in the potential file is invalid.

Illegal fix gcmc gas mass <= 0

  The computed mass of the designated gas molecule or atom type was less than or equal to zero.

Illegal fix wall/piston velocity

  The piston velocity must be positive.

Illegal integrate style

  Self-explanatory.

Illegal number of angle table entries

  There must be at least 2 table entries.

Illegal number of bond table entries

  There must be at least 2 table entries.

Illegal number of pair table entries

  There must be at least 2 table entries.

Illegal simulation box

  The lower bound of the simulation box is greater than the upper bound.

Improper atom missing in delete_bonds

  The delete_bonds command cannot find one or more atoms in a particular improper on a particular processor. The pairwise cutoff is too short or the atoms are too far apart to make a valid improper.

Improper atom missing in set command

  The set command cannot find one or more atoms in a particular improper on a particular processor. The pairwise cutoff is too short or the atoms are too far apart to make a valid improper.

Improper atoms %d %d %d %d missing on proc %d at step %ld

  One or more of 4 atoms needed to compute a particular improper are missing on this processor. Typically this is because the pairwise cutoff is set too short or the improper has blown apart and an atom is too far away.

Improper coeff for hybrid has invalid style

Improper style hybrid uses another improper style as one of its coefficients. The improper style used in the improper_coeff command or read from a restart file is not recognized.

Improper coeffs are not set

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

Improper style hybrid cannot have hybrid as an argument

Self-explanatory.

Improper style hybrid cannot have none as an argument

Self-explanatory.

Improper style hybrid cannot use same improper style twice

Self-explanatory.

Improper_coeff command before improper_style is defined

Coefficients cannot be set in the data file or via the improper_coeff command until an improper_style has been assigned.

Improper_coeff command before simulation box is defined

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

Improper_coeff command when no impropers allowed

The chosen atom style does not allow for impropers to be defined.

Improper_style command when no impropers allowed

The chosen atom style does not allow for impropers to be defined.

Impropers assigned incorrectly

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

Impropers defined but no improper types

The data file header lists improper but no improper types.

Inconsistent iparam/jparam values in fix bond/create command

If itype and jtype are the same, then their maxbond and newtype settings must also be the same.

Inconsistent line segment in data file

The end points of the line segment are not equal distances from the center point which is the atom coordinate.

Inconsistent triangle in data file

The centroid of the triangle as defined by the corner points is not the atom coordinate.

Incorrect # of floating-point values in Bodies section of data file

See doc page for body style.

Incorrect # of integer values in Bodies section of data file

See doc page for body style.

Incorrect args for angle coefficients

Self-explanatory. Check the input script or data file.

Incorrect args for bond coefficients

Self-explanatory. Check the input script or data file.

Incorrect args for dihedral coefficients

Self-explanatory. Check the input script or data file.

Incorrect args for improper coefficients

Self-explanatory. Check the input script or data file.

Incorrect args for pair coefficients

Self-explanatory. Check the input script or data file.

Incorrect args in pair_style command

Self-explanatory.

Incorrect atom format in data file

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

Incorrect bonus data format in data file

See the read_data doc page for a description of how various kinds of bonus data must be formatted for certain atom styles.

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 EwaldDisp
> Must have periodic x,y dimensions and non-periodic z dimension to use 2d slab option with Ewald.

Incorrect boundaries with slab PPPM
> Must have periodic x,y dimensions and non-periodic z dimension to use 2d slab option with PPPM.

Incorrect boundaries with slab PPPMDisp
> Must have periodic x,y dimensions and non-periodic z dimension to use 2d slab option with PPPM.

Incorrect element names in ADP potential file
> The element names in the ADP file do not match those requested.

Incorrect element names in EAM potential file
> The element names in the EAM file do not match those requested.

Incorrect format in COMB potential file
> Incorrect number of words per line in the potential file.

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

Incorrect format in NEB coordinate file
> Self-explanatory.

Incorrect format in Stillinger-Weber potential file
> Incorrect number of words per line in the potential file.

Incorrect format in TMD target file
> Format of file read by fix tmd command is incorrect.

Incorrect format in Tersoff potential file
> Incorrect number of words per line in the potential file.

Incorrect integer value in Bodies section of data file
> See doc page for body style.

Incorrect multiplicity arg for dihedral coefficients
> Self-explanatory. Check the input script or data file.

Incorrect rigid body format in fix rigid file
> The number of fields per line is not what expected.

Incorrect sign arg for dihedral coefficients
> Self-explanatory. Check the input script or data file.

Incorrect velocity format in data file
> Each atom style defines a format for the Velocity section of the data file. The read-in lines do not match.

Incorrect weight arg for dihedral coefficients
> Self-explanatory. Check the input script or data file.

Index between variable brackets must be positive
> Self-explanatory.

Indexed per-atom vector in variable formula without atom map
> Accessing a value from an atom vector requires the ability to lookup an atom index, which is provided by an atom map. An atom map does not exist (by default) for non-molecular problems. Using the atom_modify map command will force an atom map to be created.

Initial temperatures not all set in fix ttm
> Self-explantory.

Input line quote not followed by whitespace
> An end quote must be followed by whitespace.

Insertion region extends outside simulation box
> Region specified with fix pour command extends outside the global simulation box.

Insufficient Jacobi rotations for POEMS body
> Eigensolve for rigid body was not sufficiently accurate.

Insufficient Jacobi rotations for body nparticle
> Eigensolve for rigid body was not sufficiently accurate.

Insufficient Jacobi rotations for rigid body

Eigensolve for rigid body was not sufficiently accurate.

Insufficient Jacobi rotations for triangle

The calculation of the intertia tensor of the triangle failed. This should not happen if it is a reasonably shaped triangle.

Insufficient memory on accelerator

There is insufficient memory on one of the devices specified for the gpu package

Internal error in atom_style body

This error should not occur. Contact the developers.

Invalid -reorder N value

Self-explanatory.

Invalid Boolean syntax in if command

Self-explanatory.

Invalid REAX atom type

There is a mis-match between LIGGGHTS(R)-WSU atom types and the elements listed in the ReaxFF force field file.

Invalid angle style

The choice of angle style is unknown.

Invalid angle table length

Length must be 2 or greater.

Invalid angle type in Angles section of data file

Angle type must be positive integer and within range of specified angle types.

Invalid angle type index for fix shake

Self-explanatory.

Invalid args for non-hybrid pair coefficients

"NULL" is only supported in pair_coeff calls when using pair hybrid

Invalid atom ID in Angles section of data file

Atom IDs must be positive integers and within range of defined atoms.

Invalid atom ID in Atoms section of data file

Atom IDs must be positive integers.

Invalid atom ID in Bodies section of data file

Atom IDs must be positive integers and within range of defined atoms.

Invalid atom ID in Bonds section of data file

Atom IDs must be positive integers and within range of defined atoms.

Invalid atom ID in Bonus section of data file

Atom IDs must be positive integers and within range of defined atoms.

Invalid atom ID in Dihedrals section of data file

Atom IDs must be positive integers and within range of defined atoms.

Invalid atom ID in Impropers section of data file

Atom IDs must be positive integers and within range of defined atoms.

Invalid atom ID in Velocities section of data file

Atom IDs must be positive integers and within range of defined atoms.

Invalid atom mass for fix shake

Mass specified in fix shake command must be > 0.0.

Invalid atom style

The choice of atom style is unknown.

Invalid atom type in Atoms section of data file

Atom types must range from 1 to specified # of types.

Invalid atom type in create_atoms command

The create_box command specified the range of valid atom types. An invalid type is being requested.

Invalid atom type in fix bond/create command

Self-explanatory.

Invalid atom type in fix gcmc command

The atom type specified in the GCMC command does not exist.

Invalid atom type in neighbor exclusion list

Atom types must range from 1 to Ntypes inclusive.

Invalid atom type index for fix shake

Atom types must range from 1 to Ntypes inclusive.

Invalid atom types in pair_write command

Atom types must range from 1 to Ntypes inclusive.

Invalid atom vector in variable formula

The atom vector is not recognized.

Invalid atom_style body command

No body style argument was provided.

Invalid atom_style command

Self-explanatory.

Invalid attribute in dump custom command

Self-explantory.

Invalid attribute in dump local command

Self-explantory.

Invalid attribute in dump modify command

Self-explantory.

Invalid body nparticle command

Arguments in atom-style command are not correct.

Invalid body style

The choice of body style is unknown.

Invalid bond style

The choice of bond style is unknown.

Invalid bond table length

Length must be 2 or greater.

Invalid bond type in Bonds section of data file

Bond type must be positive integer and within range of specified bond types.

Invalid bond type in fix bond/break command

Self-explanatory.

Invalid bond type in fix bond/create command

Self-explanatory.

Invalid bond type index for fix shake

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

Invalid coeffs for this dihedral style

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

Invalid color in dump_modify command

The specified color name was not in the list of recognized colors. See the dump_modify doc page.

Invalid command-line argument

One or more command-line arguments is invalid. Check the syntax of the command you are using to launch LIGGGHTS(R)-WSU.

Invalid compute ID in variable formula

The compute is not recognized.

Invalid compute style

Self-explanatory.

Invalid cutoff in communicate command

Specified cutoff must be >= 0.0.

Invalid cutoffs in pair_write command

Inner cutoff must be larger than 0.0 and less than outer cutoff.

Invalid d1 or d2 value for pair colloid coeff

Neither d1 or d2 can be < 0.

Invalid data file section: Angle Coeffs

Atom style does not allow angles.

Invalid data file section: AngleAngle Coeffs

Atom style does not allow impropers.

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

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

Invalid data file section: Angles
> Atom style does not allow angles.

Invalid data file section: Bodies
> Atom style does not allow bodies.

Invalid data file section: Bond Coeffs
> Atom style does not allow bonds.

Invalid data file section: BondAngle Coeffs
> Atom style does not allow angles.

Invalid data file section: BondBond Coeffs
> Atom style does not allow angles.

Invalid data file section: BondBond13 Coeffs
> Atom style does not allow dihedrals.

Invalid data file section: Bonds
> Atom style does not allow bonds.

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

Invalid data file section: Dihedrals
> Atom style does not allow dihedrals.

Invalid data file section: Ellipsoids
> Atom style does not allow ellipsoids.

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

Invalid data file section: Improper Coeffs
> Atom style does not allow impropers.

Invalid data file section: Impropers
> Atom style does not allow impropers.

Invalid data file section: Lines
> Atom style does not allow lines.

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

Invalid data file section: Triangles
> Atom style does not allow triangles.

Invalid delta_conf in tad command
> The value must be between 0 and 1 inclusive.

Invalid density in Atoms section of data file
> Density value cannot be <= 0.0.

Invalid diameter in set command
> Self-explanatory.

Invalid dihedral style
> The choice of dihedral style is unknown.

Invalid dihedral type in Dihedrals section of data file
> Dihedral type must be positive integer and within range of specified dihedral types.

Invalid dipole length in set command
> Self-explanatory.

Invalid displace_atoms rotate axis for 2d
> Axis must be in z direction.

Invalid dump dcd filename
> Filenames used with the dump dcd style cannot be binary or compressed or cause multiple files to be written.

Invalid dump frequency

Dump frequency must be 1 or greater.

Invalid dump image element name

The specified element name was not in the standard list of elements. See the dump_modify doc page.

Invalid dump image filename

The file produced by dump image cannot be binary and must be for a single processor.

Invalid dump image persp value

Persp value must be >= 0.0.

Invalid dump image theta value

Theta must be between 0.0 and 180.0 inclusive.

Invalid dump image zoom value

Zoom value must be > 0.0.

Invalid dump reader style

Self-explanatory.

Invalid dump style

The choice of dump style is unknown.

Invalid dump xtc filename

Filenames used with the dump xtc style cannot be binary or compressed or cause multiple files to be written.

Invalid dump xyz filename

Filenames used with the dump xyz style cannot be binary or cause files to be written by each processor.

Invalid dump_modify threshhold operator

Operator keyword used for threshold specification in not recognized.

Invalid entry in -reorder file

Self-explanatory.

Invalid fix ID in variable formula

The fix is not recognized.

Invalid fix ave/time off column

Self-explantory.

Invalid fix box/relax command for a 2d simulation

Fix box/relax styles involving the z dimension cannot be used in a 2d simulation.

Invalid fix box/relax command pressure settings

If multiple dimensions are coupled, those dimensions must be specified.

Invalid fix box/relax pressure settings

Settings for coupled dimensions must be the same.

Invalid fix nvt/npt/nph command for a 2d simulation

Cannot control z dimension in a 2d model.

Invalid fix nvt/npt/nph command pressure settings

If multiple dimensions are coupled, those dimensions must be specified.

Invalid fix nvt/npt/nph pressure settings

Settings for coupled dimensions must be the same.

Invalid fix press/berendsen for a 2d simulation

The z component of pressure cannot be controlled for a 2d model.

Invalid fix press/berendsen pressure settings

Settings for coupled dimensions must be the same.

Invalid fix rigid npt/nph command for a 2d simulation

Cannot control z dimension in a 2d model.

Invalid fix rigid npt/nph command pressure settings

If multiple dimensions are coupled, those dimensions must be specified.

Invalid fix rigid npt/nph pressure settings

Settings for coupled dimensions must be the same.

Invalid fix style

The choice of fix style is unknown.

Invalid flag in force field section of restart file

Unrecognized entry in restart file.

Invalid flag in header section of restart file

Unrecognized entry in restart file.

Invalid flag in type arrays section of restart file

Unrecognized entry in restart file.

Invalid format in Bodies section of data file

The specified number of integer or floating point values does not appear.

Invalid frequency in temper command

Nevery must be > 0.

Invalid group ID in neigh_modify command

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

Invalid group function in variable formula

Group function is not recognized.

Invalid group in communicate command

Self-explanatory.

Invalid image color range

The lo value in the range is larger than the hi value.

Invalid image up vector

Up vector cannot be (0,0,0).

Invalid immediate variable

Syntax of immediate value is incorrect.

Invalid improper style

The choice of improper style is unknown.

Invalid improper type in Impropers section of data file

Improper type must be positive integer and within range of specified improper types.

Invalid index for non-body particles in compute body/local command

Only indices 1,2,3 can be used for non-body particles.

Invalid index in compute body/local command

Self-explanatory.

Invalid keyword in angle table parameters

Self-explanatory.

Invalid keyword in bond table parameters

Self-explanatory.

Invalid keyword in compute angle/local command

Self-explanatory.

Invalid keyword in compute bond/local command

Self-explanatory.

Invalid keyword in compute dihedral/local command

Self-explanatory.

Invalid keyword in compute improper/local command

Self-explanatory.

Invalid keyword in compute pair/local command

Self-explanatory.

Invalid keyword in compute property/atom command

Self-explanatory.

Invalid keyword in compute property/local command

Self-explanatory.

Invalid keyword in compute property/molecule command

Self-explanatory.

Invalid keyword in dump cfg command

Self-explanatory.

Invalid keyword in pair table parameters

Keyword used in list of table parameters is not recognized.

Invalid keyword in thermo_style custom command

One or more specified keywords are not recognized.

Invalid kspace style

The choice of kspace style is unknown.

Invalid length in set command

Self-explanatory.

Invalid mass in set command

Self-explanatory.

Invalid mass line in data file

Self-explanatory.

Invalid mass value

Self-explanatory.

Invalid math function in variable formula

Self-explanatory.

Invalid math/group/special function in variable formula

Self-explanatory.

Invalid option in lattice command for non-custom style

Certain lattice keywords are not supported unless the lattice style is "custom".

Invalid order of forces within respa levels

For respa, ordering of force computations within respa levels must obey certain rules. E.g. bonds cannot be compute less frequently than angles, pairwise forces cannot be computed less frequently than kspace, etc.

Invalid pair style

The choice of pair style is unknown.

Invalid pair table cutoff

Cutoffs in pair_coeff command are not valid with read-in pair table.

Invalid pair table length

Length of read-in pair table is invalid

Invalid partitions in processors part command

Valid partitions are numbered 1 to N and the sender and receiver cannot be the same partition.

Invalid radius in Atoms section of data file

Radius must be >= 0.0.

Invalid random number seed in fix ttm command

Random number seed must be > 0.

Invalid random number seed in set command

Random number seed must be > 0.

Invalid region style

The choice of region style is unknown.

Invalid replace values in compute reduce

Self-explanatory.

Invalid rigid body ID in fix rigid file

The ID does not match the number or an existing ID of rigid bodies that are defined by the fix rigid command.

Invalid run command N value

The number of timesteps must fit in a 32-bit integer. If you want to run for more steps than this, perform multiple shorter runs.

Invalid run command start/stop value

Self-explanatory.

Invalid run command upto value

Self-explanatory.

Invalid seed for Marsaglia random # generator

The initial seed for this random number generator must be a positive integer less than or equal to 900 million.

Invalid seed for Park random # generator

The initial seed for this random number generator must be a positive integer.

Invalid shape in Ellipsoids section of data file
> Self-explanatory.

Invalid shape in Triangles section of data file
> Two or more of the triangle corners are duplicate points.

Invalid shape in set command
> Self-explanatory.

Invalid shear direction for fix wall/gran
> Self-explanatory.

Invalid special function in variable formula
> Self-explanatory.

Invalid style in pair_write command
> Self-explanatory. Check the input script.

Invalid syntax in variable formula
> Self-explanatory.

Invalid t_event in prd command
> Self-explanatory.

Invalid t_event in tad command
> The value must be greater than 0.

Invalid thermo keyword in variable formula
> The keyword is not recognized.

Invalid tmax in tad command
> The value must be greater than 0.0.

Invalid type for mass set
> Mass command must set a type from 1-N where N is the number of atom types.

Invalid value in set command
> The value specified for the setting is invalid, likely because it is too small or too large.

Invalid variable evaluation in variable formula
> A variable used in a formula could not be evaluated.

Invalid variable in next command
> Self-explanatory.

Invalid variable in special function next
> Only file-style variables can be used with the next() function.

Invalid variable name
> Variable name used in an input script line is invalid.

Invalid variable name in variable formula
> Variable name is not recognized.

Invalid variable style with next command
> Variable styles *equal* and *world* cannot be used in a next command.

Invalid wiggle direction for fix wall/gran
> Self-explanatory.

Invoked angle equil angle on angle style none
> Self-explanatory.

Invoked angle single on angle style none
> Self-explanatory.

Invoked bond equil distance on bond style none
> Self-explanatory.

Invoked bond single on bond style none
> Self-explanatory.

Invoked pair single on pair style none
> A command (e.g. a dump) attempted to invoke the single() function on a pair style none, which is illegal. You are probably attempting to compute per-atom quantities with an undefined pair style.

KIM neighbor iterator exceeded range
> This should not happen. It likely indicates a bug in the KIM implementation of the interatomic potential where it is requesting neighbors incorrectly.

KSpace accuracy must be > 0

    The kspace accuracy designated in the input must be greater than zero.

KSpace accuracy too large to estimate G vector

    Reduce the accuracy request or specify gwald explicitly via the kspace_modify command.

KSpace accuracy too low

    Requested accuracy must be less than 1.0.

KSpace solver requires a pair style

    No pair style is defined.

KSpace style has not yet been set

    Cannot use kspace_modify command until a kspace style is set.

KSpace style is incompatible with Pair style

    Setting a kspace style requires that a pair style with a long-range Coulombic or dispersion component be used.

Keyword %s in MEAM parameter file not recognized

    Self-explanatory.

Kspace style does not support compute group/group

    Self-explanatory.

Kspace style pppm/disp/tip4p requires newton on

    Self-explanatory.

Kspace style pppm/tip4p requires newton on

    Self-explanatory.

Kspace style requires atom attribute q

    The atom style defined does not have these attributes.

Kspace style with selected options requires atom attribute q

    The atom style defined does not have these attributes. Change the atom style or switch of the coulomb solver.

LIGGGHTS(R)-WSU unit_style lj not supported by KIM models

    Self-explanatory. Check the input script or data file.

LJ6 off not supported in pair_style buck/long/coul/long

    Self-exlanatory.

Label wasn't found in input script

    Self-explanatory.

Lattice orient vectors are not orthogonal

    The three specified lattice orientation vectors must be mutually orthogonal.

Lattice orient vectors are not right-handed

    The three specified lattice orientation vectors must create a right-handed coordinate system such that a1 cross a2 = a3.

Lattice primitive vectors are collinear

    The specified lattice primitive vectors do not for a unit cell with non-zero volume.

Lattice settings are not compatible with 2d simulation

    One or more of the specified lattice vectors has a non-zero z component.

Lattice spacings are invalid

    Each x,y,z spacing must be > 0.

Lattice style incompatible with simulation dimension

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

Log of zero/negative value in variable formula

    Self-explanatory.

Lost atoms via balance: original %ld current %ld

    This should not occur. Report the problem to the developers.

Lost atoms: original %ld current %ld

    Lost atoms are checked for each time thermo output is done. See the thermo_modify lost command for options. Lost atoms usually indicate bad dynamics, e.g. atoms have been blown far out of the simulation box, or moved futher than one processor's sub-domain away before reneighboring.

MEAM library error %d

A call to the MEAM Fortran library returned an error.

MPI_LMP_BIGINT and bigint in lmptype.h are not compatible

The size of the MPI datatype does not match the size of a bigint.

MPI_LMP_TAGINT and tagint in lmptype.h are not compatible

The size of the MPI datatype does not match the size of a tagint.

MSM grid is too large

The global MSM grid is larger than OFFSET in one or more dimensions. OFFSET is currently set to 16384. You likely need to decrease the requested accuracy.

MSM order must be 4, 6, 8, or 10

This is a limitation of the MSM implementation in LIGGGHTS(R)-WSU: the MSM order can only be 4, 6, 8, or 10.

Mass command before simulation box is defined

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

Min_style command before simulation box is defined

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

Minimization could not find thermo_pe compute

This compute is created by the thermo command. It must have been explicitly deleted by a uncompute command.

Minimize command before simulation box is defined

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

Mismatched brackets in variable

Self-explanatory.

Mismatched compute in variable formula

A compute is referenced incorrectly or a compute that produces per-atom values is used in an equal-style variable formula.

Mismatched fix in variable formula

A fix is referenced incorrectly or a fix that produces per-atom values is used in an equal-style variable formula.

Mismatched variable in variable formula

A variable is referenced incorrectly or an atom-style variable that produces per-atom values is used in an equal-style variable formula.

Modulo 0 in variable formula

Self-explanatory.

Molecular data file has too many atoms

These kids of data files are currently limited to a number of atoms that fits in a 32-bit integer.

Molecule count changed in compute atom/molecule

Number of molecules must remain constant over time.

Molecule count changed in compute com/molecule

Number of molecules must remain constant over time.

Molecule count changed in compute gyration/molecule

Number of molecules must remain constant over time.

Molecule count changed in compute inertia/molecule

Number of molecules must remain constant over time.

Molecule count changed in compute msd/molecule

Number of molecules must remain constant over time.

Molecule count changed in compute property/molecule

Number of molecules must remain constant over time.

More than one fix deform

Only one fix deform can be defined at a time.

More than one fix freeze

Only one of these fixes can be defined, since the granular pair potentials access it.

More than one fix shake

Only one fix shake can be defined.

Must define angle_style before Angle Coeffs

Must use an angle_style command before reading a data file that defines Angle Coeffs.

Must define angle_style before BondAngle Coeffs

Must use an angle_style command before reading a data file that defines Angle Coeffs.

Must define angle_style before BondBond Coeffs

Must use an angle_style command before reading a data file that defines Angle Coeffs.

Must define bond_style before Bond Coeffs

Must use a bond_style command before reading a data file that defines Bond Coeffs.

Must define dihedral_style before AngleAngleTorsion Coeffs

Must use a dihedral_style command before reading a data file that defines AngleAngleTorsion Coeffs.

Must define dihedral_style before AngleTorsion Coeffs

Must use a dihedral_style command before reading a data file that defines AngleTorsion Coeffs.

Must define dihedral_style before BondBond13 Coeffs

Must use a dihedral_style command before reading a data file that defines BondBond13 Coeffs.

Must define dihedral_style before Dihedral Coeffs

Must use a dihedral_style command before reading a data file that defines Dihedral Coeffs.

Must define dihedral_style before EndBondTorsion Coeffs

Must use a dihedral_style command before reading a data file that defines EndBondTorsion Coeffs.

Must define dihedral_style before MiddleBondTorsion Coeffs

Must use a dihedral_style command before reading a data file that defines MiddleBondTorsion Coeffs.

Must define improper_style before AngleAngle Coeffs

Must use an improper_style command before reading a data file that defines AngleAngle Coeffs.

Must define improper_style before Improper Coeffs

Must use an improper_style command before reading a data file that defines Improper Coeffs.

Must define pair_style before Pair Coeffs

Must use a pair_style command before reading a data file that defines Pair Coeffs.

Must have more than one processor partition to temper

Cannot use the temper command with only one processor partition. Use the -partition command-line option.

Must read Atoms before Angles

The Atoms section of a data file must come before an Angles section.

Must read Atoms before Bodies

The Atoms section of a data file must come before a Bodies 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 Ellipsoids

The Atoms section of a data file must come before a Ellipsoids section.

Must read Atoms before Impropers

The Atoms section of a data file must come before an Impropers section.

Must read Atoms before Lines

The Atoms section of a data file must come before a Lines section.

Must read Atoms before Triangles

The Atoms section of a data file must come before a Triangles 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 shrink-wrap piston boundary

The boundary style of the face where the piston is applied must be of type s (shrink-wrapped).

Must specify a region in fix deposit

The region keyword must be specified with this fix.

Must specify a region in fix pour

The region keyword must be specified with this fix.

Must use -in switch with multiple partitions

A multi-partition simulation cannot read the input script from stdin. The -in command-line option must be used to specify a file.

Must use a block or cylinder region with fix pour

Self-explanatory.

Must use a block region with fix pour for 2d simulations

Self-explanatory.

Must use a bond style with TIP4P potential

TIP4P potentials assume bond lengths in water are constrained by a fix shake command.

Must use a molecular atom style with fix poems molecule

Self-explanatory.

Must use a z-axis cylinder with fix pour

The axis of the cylinder region used with the fix pour command must be oriented along the z dimension.

Must use an angle style with TIP4P potential

TIP4P potentials assume angles in water are constrained by a fix shake command.

Must use atom style with molecule IDs with fix bond/swap

Self-explanatory.

Must use pair_style comb with fix qeq/comb

Self-explanatory.

Must use variable energy with fix addforce

Must define an energy vartiable when applyting a dynamic force during minimization.

NEB command before simulation box is defined

Self-explanatory.

NEB requires damped dynamics minimizer

Use a different minimization style.

NEB requires use of fix neb

Self-explanatory.

NL ramp in wall/piston only implemented in zlo for now

The ramp keyword can only be used for piston applied to face zlo.

Needed bonus data not in data file

Some atom styles require bonus data. See the read_data doc page for details.

Needed topology not in data file

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

Neigh_modify exclude molecule requires atom attribute molecule

Self-explanatory.

Neigh_modify include group != atom_modify first group

Self-explanatory.

Neighbor delay must be 0 or multiple of every setting

The delay and every parameters set via the neigh_modify command are inconsistent. If the delay setting is non-zero, then it must be a multiple of the every setting.

Neighbor include group not allowed with ghost neighbors

This is a current restriction within LIGGGHTS(R)-WSU.

Neighbor list overflow, boost neigh_modify one

There are too many neighbors of a single atom. Use the neigh_modify command to increase the max number of neighbors allowed for one atom. You may also want to boost the page size.

Neighbor list overflow, boost neigh_modify one or page

There are too many neighbors of a single atom. Use the neigh_modify command to increase the neighbor page size and the max number of neighbors allowed for one atom.

Neighbor multi not yet enabled for ghost neighbors

This is a current restriction within LIGGGHTS(R)-WSU.

Neighbor multi not yet enabled for granular

Self-explanatory.

Neighbor multi not yet enabled for rRESPA

Self-explanatory.

Neighbor page size must be >= 10x the one atom setting

This is required to prevent wasting too much memory.

New bond exceeded bonds per atom in fix bond/create

See the read_data command for info on setting the "extra bond per atom" header value to allow for additional bonds to be formed.

New bond exceeded special list size in fix bond/create

See the special_bonds extra command for info on how to leave space in the special bonds list to allow for additional bonds to be formed.

Newton bond change after simulation box is defined

The newton command cannot be used to change the newton bond value after a read_data, read_restart, or create_box command.

No Kspace style defined for compute group/group

Self-explanatory.

No OpenMP support compiled in

An OpenMP flag is set, but LIGGGHTS(R)-WSU was not built with OpenMP support.

No angle style is defined for compute angle/local

Self-explanatory.

No angles allowed with this atom style

Self-explanatory. Check data file.

No atoms in data file

The header of the data file indicated that atoms would be included, but they were not present.

No basis atoms in lattice

Basis atoms must be defined for lattice style user.

No bodies allowed with this atom style

Self-explanatory. Check data file.

No bond style is defined for compute bond/local

Self-explanatory.

No bonds allowed with this atom style

Self-explanatory. Check data file.

No box information in dump. You have to use 'box no'

Self-explanatory.

No dihedral style is defined for compute dihedral/local

Self-explanatory.

No dihedrals allowed with this atom style

Self-explanatory. Check data file.

No dump custom arguments specified

The dump custom command requires that atom quantities be specified to output to dump file.

No dump local arguments specified

Self-explanatory.

No ellipsoids allowed with this atom style

Self-explanatory. Check data file.

No fix gravity defined for fix pour

Cannot add poured particles without gravity to move them.

No improper style is defined for compute improper/local

Self-explanatory.

No impropers allowed with this atom style

Self-explanatory. Check data file.

No lines allowed with this atom style

Self-explanatory. Check data file.

No matching element in ADP potential file

The ADP potential file does not contain elements that match the requested elements.

No matching element in EAM potential file
> The EAM potential file does not contain elements that match the requested elements.

No overlap of box and region for create_atoms
> Self-explanatory.

No pair hbond/dreiding coefficients set
> Self-explanatory.

No pair style defined for compute group/group
> Cannot calculate group interactions without a pair style defined.

No pair style is defined for compute pair/local
> Self-explanatory.

No pair style is defined for compute property/local
> Self-explanatory.

No rigid bodies defined
> The fix specification did not end up defining any rigid bodies.

No triangles allowed with this atom style
> Self-explanatory. Check data file.

Non digit character between brackets in variable
> Self-explantory.

Non integer # of swaps in temper command
> Swap frequency in temper command must evenly divide the total # of timesteps.

Nprocs not a multiple of N for -reorder
> Self-explanatory.

Numeric index is out of bounds
> A command with an argument that specifies an integer or range of integers is using a value that is less than 1 or greater than the maximum allowed limit.

One or more atoms belong to multiple rigid bodies
> Two or more rigid bodies defined by the fix rigid command cannot contain the same atom.

One or zero atoms in rigid body
> Any rigid body defined by the fix rigid command must contain 2 or more atoms.

Only one cutoff allowed when requesting all long
> Self-explanatory.

Only zhi currently implemented for fix append/atoms
> Self-explanatory.

Out of range atoms - cannot compute MSM
> One or more atoms are attempting to map their charge to a MSM grid point that is not owned by a processor. This is likely for one of two reasons, both of them bad. First, it may mean that an atom near the boundary of a processor's sub-domain has moved more than 1/2 the neighbor skin distance without neighbor lists being rebuilt and atoms being migrated to new processors. This also means you may be missing pairwise interactions that need to be computed. The solution is to change the re-neighboring criteria via the neigh_modify command. The safest settings are "delay 0 every 1 check yes". Second, it may mean that an atom has moved far outside a processor's sub-domain or even the entire simulation box. This indicates bad physics, e.g. due to highly overlapping atoms, too large a timestep, etc.

Out of range atoms - cannot compute PPPM
> One or more atoms are attempting to map their charge to a PPPM grid point that is not owned by a processor. This is likely for one of two reasons, both of them bad. First, it may mean that an atom near the boundary of a processor's sub-domain has moved more than 1/2 the neighbor skin distance without neighbor lists being rebuilt and atoms being migrated to new processors. This also means you may be missing pairwise interactions that need to be computed. The solution is to change the re-neighboring criteria via the neigh_modify command. The safest settings are "delay 0 every 1 check yes". Second, it may mean that an atom has moved far outside a processor's sub-domain or even the entire simulation box. This indicates bad physics, e.g. due to highly overlapping atoms, too large a timestep, etc.

Out of range atoms - cannot compute PPPMDisp

One or more atoms are attempting to map their charge to a PPPM grid point that is not owned by a processor. This is likely for one of two reasons, both of them bad. First, it may mean that an atom near the boundary of a processor's sub-domain has moved more than 1/2 the [neighbor skin distance](#) without neighbor lists being rebuilt and atoms being migrated to new processors. This also means you may be missing pairwise interactions that need to be computed. The solution is to change the re-neighboring criteria via the [neigh_modify](#) command. The safest settings are "delay 0 every 1 check yes". Second, it may mean that an atom has moved far outside a processor's sub-domain or even the entire simulation box. This indicates bad physics, e.g. due to highly overlapping atoms, too large a timestep, etc.

Overlapping large/large in pair colloid
This potential is infinite when there is an overlap.

Overlapping small/large in pair colloid
This potential is infinite when there is an overlap.

POEMS fix must come before NPT/NPH fix
NPT/NPH fix must be defined in input script after all poems fixes, else the fix contribution to the pressure virial is incorrect.

PPPM grid is too large
The global PPPM grid is larger than OFFSET in one or more dimensions. OFFSET is currently set to 4096. You likely need to decrease the requested accuracy.

PPPM grid stencil extends beyond nearest neighbor processor
This is not allowed if the kspace_modify overlap setting is no.

PPPM order < minimum allowed order
The default minimum order is 2. This can be reset by the kspace_modify minorder command.

PPPM order cannot be < 2 or > than %d
This is a limitation of the PPPM implementation in LIGGGHTS(R)-WSU.

PPPMDisp Coulomb grid is too large
The global PPPM grid is larger than OFFSET in one or more dimensions. OFFSET is currently set to 4096. You likely need to decrease the requested accuracy.

PPPMDisp Dispersion grid is too large
The global dispersion grid is larger than OFFSET in one or more dimensions. OFFSET is currently set to 4096. You likely need to decrease the requested accuracy.

PPPMDisp coulomb order cannot be greater than %d
This is a limitation of the PPPM implementation in LIGGGHTS(R)-WSU.

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

PRD nsteps must be multiple of t_event
Self-explanatory.

PRD t_corr must be multiple of t_event
Self-explanatory.

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

Package cuda command without USER-CUDA installed
The USER-CUDA package must be installed via "make yes-user-cuda" before LIGGGHTS(R)-WSU is built.

Pair body requires atom style body
Self-explanatory.

Pair body requires body style nparticle
This pair style is specific to the nparticle body style.

Pair brownian requires atom style sphere
Self-explanatory.

Pair brownian requires extended particles
One of the particles has radius 0.0.

Pair brownian requires monodisperse particles
All particles must be the same finite size.

Pair brownian/poly requires atom style sphere
Self-explanatory.
Pair brownian/poly requires extended particles
One of the particles has radius 0.0.
Pair brownian/poly requires newton pair off
Self-explanatory.
Pair coeff for hybrid has invalid style
Style in pair coeff must have been listed in pair_style command.
Pair colloid/poly requires atom style sphere
Self-explanatory.
Pair coul/wolf requires atom attribute q
The atom style defined does not have this attribute.
Pair cutoff < Respa interior cutoff
One or more pairwise cutoffs are too short to use with the specified rRESPA cutoffs.
Pair dipole/cut requires atom attributes q, mu, torque
The atom style defined does not have these attributes.
Pair dipole/cut/gpu requires atom attributes q, mu, torque
The atom style defined does not have this attribute.
Pair distance < table inner cutoff
Two atoms are closer together than the pairwise table allows.
Pair distance > table outer cutoff
Two atoms are further apart than the pairwise table allows.
Pair dpd requires ghost atoms store velocity
Use the communicate vel yes command to enable this.
Pair gayberne epsilon a,b,c coeffs are not all set
Each atom type involved in pair_style gayberne must have these 3 coefficients set at least once.
Pair gayberne requires atom style ellipsoid
Self-explanatory.
Pair gayberne requires atoms with same type have same shape
Self-explanatory.
Pair gayberne/gpu requires atom style ellipsoid
Self-explanatory.
Pair gayberne/gpu requires atoms with same type have same shape
Self-explanatory.
Pair granular requires atom style sphere
Self-explanatory.
Pair granular requires ghost atoms store velocity
Use the communicate vel yes command to enable this.
Pair granular with shear history requires newton pair off
This is a current restriction of the implementation of pair granular styles with history.
Pair hybrid sub-style does not support single call
You are attempting to invoke a single() call on a pair style that doesn't support it.
Pair hybrid sub-style is not used
No pair_coeff command used a sub-style specified in the pair_style command.
Pair inner cutoff < Respa interior cutoff
One or more pairwise cutoffs are too short to use with the specified rRESPA cutoffs.
Pair inner cutoff >= Pair outer cutoff
The specified cutoffs for the pair style are inconsistent.
Pair line/lj requires atom style line
Self-explanatory.
Pair lubricate requires atom style sphere
Self-explanatory.
Pair lubricate requires ghost atoms store velocity
Use the communicate vel yes command to enable this.

Pair lubricate requires monodisperse particles
>All particles must be the same finite size.

Pair lubricate/poly requires atom style sphere
>Self-explanatory.

Pair lubricate/poly requires extended particles
>One of the particles has radius 0.0.

Pair lubricate/poly requires ghost atoms store velocity
>Use the communicate vel yes command to enable this.

Pair lubricate/poly requires newton pair off
>Self-explanatory.

Pair lubricateU requires atom style sphere
>Self-explanatory.

Pair lubricateU requires ghost atoms store velocity
>Use the communicate vel yes command to enable this.

Pair lubricateU requires monodisperse particles
>All particles must be the same finite size.

Pair lubricateU/poly requires ghost atoms store velocity
>Use the communicate vel yes command to enable this.

Pair lubricateU/poly requires newton pair off
>Self-explanatory.

Pair peri lattice is not identical in x, y, and z
>The lattice defined by the lattice command must be cubic.

Pair peri requires a lattice be defined
>Use the lattice command for this purpose.

Pair peri requires an atom map, see atom_modify
>Even for atomic systems, an atom map is required to find Peridynamic bonds. Use the atom_modify command to define one.

Pair resquared epsilon a,b,c coeffs are not all set
>Self-explanatory.

Pair resquared epsilon and sigma coeffs are not all set
>Self-explanatory.

Pair resquared requires atom style ellipsoid
>Self-explanatory.

Pair resquared requires atoms with same type have same shape
>Self-explanatory.

Pair resquared/gpu requires atom style ellipsoid
>Self-explanatory.

Pair resquared/gpu requires atoms with same type have same shape
>Self-explanatory.

Pair style AIREBO requires atom IDs
>This is a requirement to use the AIREBO potential.

Pair style AIREBO requires newton pair on
>See the newton command. This is a restriction to use the AIREBO potential.

Pair style BOP requires atom IDs
>This is a requirement to use the BOP potential.

Pair style BOP requires newton pair on
>See the newton command. This is a restriction to use the BOP potential.

Pair style COMB requires atom IDs
>This is a requirement to use the AIREBO potential.

Pair style COMB requires atom attribute q
>Self-explanatory.

Pair style COMB requires newton pair on
>See the newton command. This is a restriction to use the COMB potential.

Pair style LCBOP requires atom IDs

This is a requirement to use the LCBOP potential.

Pair style LCBOP requires newton pair on
> See the newton command. This is a restriction to use the LCBOP potential.

Pair style MEAM requires newton pair on
> See the newton command. This is a restriction to use the MEAM potential.

Pair style Stillinger-Weber requires atom IDs
> This is a requirement to use the SW potential.

Pair style Stillinger-Weber requires newton pair on
> See the newton command. This is a restriction to use the SW potential.

Pair style Tersoff requires atom IDs
> This is a requirement to use the Tersoff potential.

Pair style Tersoff requires newton pair on
> See the newton command. This is a restriction to use the Tersoff potential.

Pair style bop requires comm ghost cutoff at least 3x larger than %g
> Use the communicate ghost command to set this. See the pair bop doc page for more details.

Pair style born/coul/long requires atom attribute q
> An atom style that defines this attribute must be used.

Pair style born/coul/long/gpu requires atom attribute q
> The atom style defined does not have this attribute.

Pair style born/coul/wolf requires atom attribute q
> The atom style defined does not have this attribute.

Pair style buck/coul/cut requires atom attribute q
> The atom style defined does not have this attribute.

Pair style buck/coul/long requires atom attribute q
> The atom style defined does not have these attributes.

Pair style buck/coul/long/gpu requires atom attribute q
> The atom style defined does not have this attribute.

Pair style buck/long/coul/long requires atom attribute q
> The atom style defined does not have this attribute.

Pair style coul/cut requires atom attribute q
> The atom style defined does not have these attributes.

Pair style coul/dsf requires atom attribute q
> The atom style defined does not have this attribute.

Pair style coul/dsf/gpu requires atom attribute q
> The atom style defined does not have this attribute.

Pair style coul/long/gpu requires atom attribute q
> The atom style defined does not have these attributes.

Pair style does not have extra field requested by compute pair/local
> The pair style does not support the pN value requested by the compute pair/local command.

Pair style does not support bond_style quartic
> The pair style does not have a single() function, so it can not be invoked by bond_style quartic.

Pair style does not support compute group/group
> The pair_style does not have a single() function, so it cannot be invokded by the compute group/group command.

Pair style does not support compute pair/local
> The pair style does not have a single() function, so it can not be invoked by compute pair/local.

Pair style does not support compute property/local
> The pair style does not have a single() function, so it can not be invoked by fix bond/swap.

Pair style does not support fix bond/swap
> The pair style does not have a single() function, so it can not be invoked by fix bond/swap.

Pair style does not support pair_write
> The pair style does not have a single() function, so it can not be invoked by pair write.

Pair style does not support rRESPA inner/middle/outer
> You are attempting to use rRESPA options with a pair style that does not support them.

Pair style granular with history requires atoms have IDs
> Atoms in the simulation do not have IDs, so history effects cannot be tracked by the granular pair potential.

Pair style hbond/dreiding requires an atom map, see atom_modify
> Self-explanatory.

Pair style hbond/dreiding requires atom IDs
> Self-explanatory.

Pair style hbond/dreiding requires molecular system
> Self-explanatory.

Pair style hbond/dreiding requires newton pair on
> See the newton command for details.

Pair style hybrid cannot have hybrid as an argument
> Self-explanatory.

Pair style hybrid cannot have none as an argument
> Self-explanatory.

Pair style is incompatible with KSpace style
> If a pair style with a long-range Coulombic component is selected, then a kspace style must also be used.

Pair style lj/charmm/coul/charmm requires atom attribute q
> The atom style defined does not have these attributes.

Pair style lj/charmm/coul/long requires atom attribute q
> The atom style defined does not have these attributes.

Pair style lj/charmm/coul/long/gpu requires atom attribute q
> The atom style defined does not have this attribute.

Pair style lj/class2/coul/cut requires atom attribute q
> The atom style defined does not have this attribute.

Pair style lj/class2/coul/long requires atom attribute q
> The atom style defined does not have this attribute.

Pair style lj/class2/coul/long/gpu requires atom attribute q
> The atom style defined does not have this attribute.

Pair style lj/cut/coul/cut requires atom attribute q
> The atom style defined does not have this attribute.

Pair style lj/cut/coul/cut/gpu requires atom attribute q
> The atom style defined does not have this attribute.

Pair style lj/cut/coul/debye/gpu requires atom attribute q
> The atom style defined does not have this attribute.

Pair style lj/cut/coul/dsf requires atom attribute q
> The atom style defined does not have these attributes.

Pair style lj/cut/coul/dsf/gpu requires atom attribute q
> The atom style defined does not have this attribute.

Pair style lj/cut/coul/long requires atom attribute q
> The atom style defined does not have this attribute.

Pair style lj/cut/coul/long/gpu requires atom attribute q
> The atom style defined does not have this attribute.

Pair style lj/cut/tip4p/long requires atom IDs
> There are no atom IDs defined in the system and the TIP4P potential requires them to find O,H atoms with a water molecule.

Pair style lj/cut/tip4p/long requires atom attribute q
> The atom style defined does not have these attributes.

Pair style lj/cut/tip4p/long requires newton pair on
> This is because the computation of constraint forces within a water molecule adds forces to atoms owned by other processors.

Pair style lj/cut/coul/msm requires atom attribute q
> The atom style defined does not have this attribute.

Pair style lj/gromacs/coul/gromacs requires atom attribute q
> An atom_style with this attribute is needed.

Pair style lj/long/coul/long requires atom attribute q
> The atom style defined does not have this attribute.

Pair style lj/long/tip4p/long requires atom IDs
> There are no atom IDs defined in the system and the TIP4P potential requires them to find O,H atoms with a water molecule.

Pair style lj/long/tip4p/long requires atom attribute q
> The atom style defined does not have these attributes.

Pair style lj/long/tip4p/long requires newton pair on
> This is because the computation of constraint forces within a water molecule adds forces to atoms owned by other processors.

Pair style peri requires atom style peri
> Self-explanatory.

Pair style reax requires atom IDs
> This is a requirement to use the ReaxFF potential.

Pair style reax requires newton pair on
> This is a requirement to use the ReaxFF potential.

Pair style requires a KSpace style
> No kspace style is defined.

Pair table cutoffs must all be equal to use with KSpace
> When using pair style table with a long-range KSpace solver, the cutoffs for all atom type pairs must all be the same, since the long-range solver starts at that cutoff.

Pair table parameters did not set N
> List of pair table parameters must include N setting.

Pair tersoff/zbl requires metal or real units
> This is a current restriction of this pair potential.

Pair tri/lj requires atom style tri
> Self-explanatory.

Pair yukawa/colloid requires atom style sphere
> Self-explantory.

Pair yukawa/colloid requires atoms with same type have same radius
> Self-explantory.

Pair yukawa/colloid/gpu requires atom style sphere
> Self-explanatory.

PairKIM only works with 3D problems.
> This is a current restriction of this pair style.

Pair_coeff command before pair_style is defined
> Self-explanatory.

Pair_coeff command before simulation box is defined
> The pair_coeff command cannot be used before a read_data, read_restart, or create_box command.

Pair_modify command before pair_style is defined
> Self-explanatory.

Pair_write command before pair_style is defined
> Self-explanatory.

Particle on or inside fix wall surface
> Particles must be "exterior" to the wall in order for energy/force to be calculated.

Particle on or inside surface of region used in fix wall/region
> Particles must be "exterior" to the region surface in order for energy/force to be calculated.

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

Per-atom energy was not tallied on needed timestep
> You are using a thermo keyword that requires potentials to have tallied energy, but they didn't on this timestep. See the variable doc page for ideas on how to make this work.

Per-atom fix in equal-style variable formula
  Equal-style variables cannot use per-atom quantities.
Per-atom virial was not tallied on needed timestep
  You are using a thermo keyword that requires potentials to have tallied the virial, but they didn't on this timestep. See the variable doc page for ideas on how to make this work.
Per-processor system is too big
  The number of owned atoms plus ghost atoms on a single processor must fit in 32-bit integer.
Potential energy ID for fix neb does not exist
  Self-explanatory.
Potential energy ID for fix nvt/nph/npt does not exist
  A compute for potential energy must be defined.
Potential file has duplicate entry
  The potential file for a SW or Tersoff potential has more than one entry for the same 3 ordered elements.
Potential file is missing an entry
  The potential file for a SW or Tersoff potential does not have a needed entry.
Power by 0 in variable formula
  Self-explanatory.
Pressure ID for fix box/relax does not exist
  The compute ID needed to compute pressure for the fix does not exist.
Pressure ID for fix modify does not exist
  Self-explanatory.
Pressure ID for fix npt/nph does not exist
  Self-explanatory.
Pressure ID for fix press/berendsen does not exist
  The compute ID needed to compute pressure for the fix does not exist.
Pressure ID for fix rigid npt/nph does not exist
  Self-explanatory.
Pressure ID for thermo does not exist
  The compute ID needed to compute pressure for thermodynamics does not exist.
Pressure control can not be used with fix nvt
  Self-explanatory.
Pressure control can not be used with fix nvt/asphere
  Self-explanatory.
Pressure control can not be used with fix nvt/sllod
  Self-explanatory.
Pressure control can not be used with fix nvt/sphere
  Self-explanatory.
Pressure control must be used with fix nph
  Self-explanatory.
Pressure control must be used with fix nph/asphere
  Self-explanatory.
Pressure control must be used with fix nph/sphere
  Self-explanatory.
Pressure control must be used with fix nphug
  A pressure control keyword (iso, aniso, tri, x, y, or z) must be provided.
Pressure control must be used with fix npt
  Self-explanatory.
Pressure control must be used with fix npt/asphere
  Self-explanatory.
Pressure control must be used with fix npt/sphere
  Self-explanatory.
Processor count in z must be 1 for 2d simulation
  Self-explanatory.

Processor partitions are inconsistent

      The total number of processors in all partitions must match the number of processors LIGGGHTS(R)-WSU 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.

Processors custom grid file is inconsistent

      The vales in the custom file are not consistent with the number of processors you are running on or the Px,Py,Pz settings of the processors command. Or there was not a setting for every processor.

Processors grid numa and map style are incompatible

      Using numa for gstyle in the processors command requires using cart for the map option.

Processors part option and grid style are incompatible

      Cannot use gstyle numa or custom with the part option.

Processors twogrid requires proc count be a multiple of core count

      Self-explanatory.

Pstart and Pstop must have the same value

      Self-explanatory.

R0 < 0 for fix spring command

      Equilibrium spring length is invalid.

Read_dump command before simulation box is defined

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

Read_dump field not found in dump file

      Self-explanatory.

Read_dump triclinic status does not match simulation

      Both the dump snapshot and the current LIGGGHTS(R)-WSU simulation must be using either an orthogonal or triclinic box.

Read_dump x,y,z fields do not have consistent scaling

      Self-explanatory.

Reax_defs.h setting for NATDEF is too small

      Edit the setting in the ReaxFF library and re-compile the library and re-build LIGGGHTS(R)-WSU.

Reax_defs.h setting for NNEIGHMAXDEF is too small

      Edit the setting in the ReaxFF library and re-compile the library and re-build LIGGGHTS(R)-WSU.

Receiving partition in processors part command is already a receiver

      Cannot specify a partition to be a receiver twice.

Region ID for compute reduce/region does not exist

      Self-explanatory.

Region ID for compute temp/region does not exist

      Self-explanatory.

Region ID for dump custom does not exist

      Self-explanatory.

Region ID for fix addforce does not exist

      Self-explanatory.

Region ID for fix ave/spatial does not exist

      Self-explanatory.

Region ID for fix aveforce does not exist

      Self-explanatory.

Region ID for fix deposit does not exist

      Self-explanatory.

Region ID for fix evaporate does not exist

      Self-explanatory.

Region ID for fix gcmc does not exist

      Self-explanatory.

Region ID for fix heat does not exist

      Self-explanatory.

Region ID for fix setforce does not exist

Self-explanatory.

Region ID for fix wall/region does not exist

Self-explanatory.

Region ID in variable formula does not exist

Self-explanatory.

Region cannot have 0 length rotation vector

Self-explanatory.

Region intersect region ID does not exist

Self-explanatory.

Region union or intersect cannot be dynamic

The sub-regions can be dynamic, but not the combined region.

Region union region ID does not exist

One or more of the region IDs specified by the region union command does not exist.

Replacing a fix, but new style != old style

A fix ID can be used a 2nd time, but only if the style matches the previous fix. In this case it is assumed you with to reset a fix's parameters. This error may mean you are mistakenly re-using a fix ID when you do not intend to.

Replicate command before simulation box is defined

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

Replicate did not assign all atoms correctly

Atoms replicated by the replicate command were not assigned correctly to processors. This is likely due to some atom coordinates being outside a non-periodic simulation box.

Replicated molecular system atom IDs are too big

See the setting for the allowed atom ID size in the src/lmptype.h file.

Replicated system is too big

See the setting for bigint in the src/lmptype.h file.

Rerun command before simulation box is defined

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

Rerun dump file does not contain requested snapshot

Self-explanatory.

Resetting timestep is not allowed with fix move

This is because fix move is moving atoms based on elapsed time.

Respa inner cutoffs are invalid

The first cutoff must be <= the second cutoff.

Respa levels must be >= 1

Self-explanatory.

Respa middle cutoffs are invalid

The first cutoff must be <= the second cutoff.

Restart variable returned a bad timestep

The variable must return a timestep greater than the current timestep.

Restrain atoms %d %d %d %d missing on proc %d at step %ld

The 4 atoms in a restrain dihedral specified by the fix restrain command are not all accessible to a processor. This probably means an atom has moved too far.

Restrain atoms %d %d %d missing on proc %d at step %ld

The 3 atoms in a restrain angle specified by the fix restrain command are not all accessible to a processor. This probably means an atom has moved too far.

Restrain atoms %d %d missing on proc %d at step %ld

The 2 atoms in a restrain bond specified by the fix restrain command are not all accessible to a processor. This probably means an atom has moved too far.

Reuse of compute ID

A compute ID cannot be used twice.

Reuse of dump ID

A dump ID cannot be used twice.

Reuse of region ID

A region ID cannot be used twice.

Rigid body atoms %d %d missing on proc %d at step %ld

This means that an atom cannot find the atom that owns the rigid body it is part of, or vice versa. The solution is to use the communicate cutoff command to insure ghost atoms are acquired from far enough away to encompass the max distance printed when the fix rigid/small command was invoked.

Rigid body has degenerate moment of inertia

Fix poems will only work with bodies (collections of atoms) that have non-zero principal moments of inertia. This means they must be 3 or more non-collinear atoms, even with joint atoms removed.

Rigid fix must come before NPT/NPH fix

NPT/NPH fix must be defined in input script after all rigid fixes, else the rigid fix contribution to the pressure virial is incorrect.

Rmask function in equal-style variable formula

Rmask is per-atom operation.

Run command before simulation box is defined

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

Run command start value is after start of run

Self-explanatory.

Run command stop value is before end of run

Self-explanatory.

Run_style command before simulation box is defined

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

SRD bin size for fix srd differs from user request

Fix SRD had to adjust the bin size to fit the simulation box. See the cubic keyword if you want this message to be an error vs warning.

SRD bins for fix srd are not cubic enough

The bin shape is not within tolerance of cubic. See the cubic keyword if you want this message to be an error vs warning.

SRD particle %d started inside big particle %d on step %ld bounce %d

See the inside keyword if you want this message to be an error vs warning.

Same dimension twice in fix ave/spatial

Self-explanatory.

Sending partition in processors part command is already a sender

Cannot specify a partition to be a sender twice.

Set command before simulation box is defined

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

Set command with no atoms existing

No atoms are yet defined so the set command cannot be used.

Set region ID does not exist

Region ID specified in set command does not exist.

Shake angles have different bond types

All 3-atom angle-constrained SHAKE clusters specified by the fix shake command that are the same angle type, must also have the same bond types for the 2 bonds in the angle.

Shake atoms %d %d %d %d missing on proc %d at step %ld

The 4 atoms in a single shake cluster specified by the fix shake command are not all accessible to a processor. This probably means an atom has moved too far.

Shake atoms %d %d %d missing on proc %d at step %ld

The 3 atoms in a single shake cluster specified by the fix shake command are not all accessible to a processor. This probably means an atom has moved too far.

Shake atoms %d %d missing on proc %d at step %ld

The 2 atoms in a single shake cluster specified by the fix shake command are not all accessible to a processor. This probably means an atom has moved too far.

Shake cluster of more than 4 atoms

A single cluster specified by the fix shake command can have no more than 4 atoms.

Shake clusters are connected

A single cluster specified by the fix shake command must have a single central atom with up to 3 other atoms bonded to it.

Shake determinant = 0.0

The determinant of the matrix being solved for a single cluster specified by the fix shake command is numerically invalid.

Shake fix must come before NPT/NPH fix

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

Small, tag, big integers are not sized correctly

See description of these 3 data types in src/lmptype.h.

Smallint setting in lmptype.h is invalid

It has to be the size of an integer.

Smallint setting in lmptype.h is not compatible

Smallint stored in restart file is not consistent with LIGGGHTS(R)-WSU version you are running.

Specified processors != physical processors

The 3d grid of processors defined by the processors command does not match the number of processors LIGGGHTS(R)-WSU is being run on.

Specified target stress must be uniaxial or hydrostatic

Self-explanatory.

Sqrt of negative value in variable formula

Self-explanatory.

Substitution for illegal variable

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

System in data file is too big

See the setting for bigint in the src/lmptype.h file.

System is not charge neutral, net charge = %g

The total charge on all atoms on the system is not 0.0, which is not valid for the long-range Coulombic solvers.

TAD nsteps must be multiple of t_event

Self-explanatory.

TIP4P hydrogen has incorrect atom type

The TIP4P pairwise computation found an H atom whose type does not agree with the specified H type.

TIP4P hydrogen is missing

The TIP4P pairwise computation failed to find the correct H atom within a water molecule.

TMD target file did not list all group atoms

The target file for the fix tmd command did not list all atoms in the fix group.

Tad command before simulation box is defined

Self-explanatory.

Tagint setting in lmptype.h is invalid

Tagint must be as large or larger than smallint.

Tagint setting in lmptype.h is not compatible

Smallint stored in restart file is not consistent with LIGGGHTS(R)-WSU version you are running.

Target temperature for fix nvt/npt/nph cannot be 0.0

Self-explanatory.

Target temperature for fix rigid/npt cannot be 0.0

Self-explanatory.

Target temperature for fix rigid/nvt cannot be 0.0

Self-explanatory.

Temper command before simulation box is defined

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

Temperature ID for fix bond/swap does not exist

Self-explanatory.

Temperature ID for fix box/relax does not exist

Self-explanatory.
Temperature ID for fix nvt/npt does not exist
>Self-explanatory.

Temperature ID for fix press/berendsen does not exist
>Self-explanatory.

Temperature ID for fix rigid nvt/npt/nph does not exist
>Self-explanatory.

Temperature ID for fix temp/berendsen does not exist
>Self-explanatory.

Temperature ID for fix temp/rescale does not exist
>Self-explanatory.

Temperature control can not be used with fix nph
>Self-explanatory.

Temperature control can not be used with fix nph/asphere
>Self-explanatory.

Temperature control can not be used with fix nph/sphere
>Self-explanatory.

Temperature control must be used with fix nphug
>The temp keyword must be provided.

Temperature control must be used with fix npt
>Self-explanatory.

Temperature control must be used with fix npt/asphere
>Self-explanatory.

Temperature control must be used with fix npt/sphere
>Self-explanatory.

Temperature control must be used with fix nvt
>Self-explanatory.

Temperature control must be used with fix nvt/asphere
>Self-explanatory.

Temperature control must be used with fix nvt/sllod
>Self-explanatory.

Temperature control must be used with fix nvt/sphere
>Self-explanatory.

Temperature for fix nvt/sllod does not have a bias
>The specified compute must compute temperature with a bias.

Tempering could not find thermo_pe compute
>This compute is created by the thermo command. It must have been explicitly deleted by a uncompute command.

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

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

Test_descriptor_string already allocated
>This should not happen. It likely indicates a bug in the pair_kim implementation.

The package gpu command is required for gpu styles
>Self-explanatory.

Thermo and fix not computed at compatible times
>Fixes generate values on specific timesteps. The thermo output does not match these timesteps.

Thermo compute array is accessed out-of-range
>Self-explanatory.

Thermo compute does not compute array
>Self-explanatory.

Thermo compute does not compute scalar
>Self-explanatory.

Thermo compute does not compute vector
> Self-explanatory.

Thermo compute vector is accessed out-of-range
> Self-explanatory.

Thermo custom variable cannot be indexed
> Self-explanatory.

Thermo custom variable is not equal-style variable
> Only equal-style variables can be output with thermodynamics, not atom-style variables.

Thermo every variable returned a bad timestep
> The variable must return a timestep greater than the current timestep.

Thermo fix array is accessed out-of-range
> Self-explanatory.

Thermo fix does not compute array
> Self-explanatory.

Thermo fix does not compute scalar
> Self-explanatory.

Thermo fix does not compute vector
> Self-explanatory.

Thermo fix vector is accessed out-of-range
> Self-explanatory.

Thermo keyword in variable requires lattice be defined
> The xlat, ylat, zlat keywords refer to lattice properties.

Thermo keyword in variable requires thermo to use/init pe
> You are using a thermo keyword in a variable that requires potential energy to be calculated, but your thermo output does not use it. Add it to your thermo output.

Thermo keyword in variable requires thermo to use/init press
> You are using a thermo keyword in a variable that requires pressure to be calculated, but your thermo output does not use it. Add it to your thermo output.

Thermo keyword in variable requires thermo to use/init temp
> You are using a thermo keyword in a variable that requires temperature to be calculated, but your thermo output does not use it. Add it to your thermo output.

Thermo keyword requires lattice be defined
> The xlat, ylat, zlat keywords refer to lattice properties.

Thermo style does not use press
> Cannot use thermo_modify to set this parameter since the thermo_style is not computing this quantity.

Thermo style does not use temp
> Cannot use thermo_modify to set this parameter since the thermo_style is not computing this quantity.

Thermo_modify int format does not contain d character
> Self-explanatory.

Thermo_modify pressure ID does not compute pressure
> The specified compute ID does not compute pressure.

Thermo_modify temperature ID does not compute temperature
> The specified compute ID does not compute temperature.

Thermo_style command before simulation box is defined
> The thermo_style command cannot be used before a read_data, read_restart, or create_box command.

This variable thermo keyword cannot be used between runs
> Keywords that refer to time (such as cpu, elapsed) do not make sense in between runs.

Threshhold for an atom property that isn't allocated
> A dump threshhold has been requested on a quantity that is not defined by the atom style used in this simulation.

Timestep must be >= 0
> Specified timestep is invalid.

Too big a problem to use velocity create loop all
> The system size must fit in a 32-bit integer to use this option.

Too big a timestep

>   Specified timestep is too large.

Too big a timestep for dump dcd

>   The timestep must fit in a 32-bit integer to use this dump style.

Too big a timestep for dump xtc

>   The timestep must fit in a 32-bit integer to use this dump style.

Too few bits for lookup table

>   Table size specified via pair_modify command does not work with your machine's floating point representation.

Too many MSM grid levels

>   The max number of MSM grid levels is hardwired to 10.

Too many atom pairs for pair bop

>   The number of atomic pairs exceeds the expected number. Check your atomic structure to ensure that it is realistic.

Too many atom sorting bins

>   This is likely due to an immense simulation box that has blown up to a large size.

Too many atom triplets for pair bop

>   The number of three atom groups for angle determinations exceeds the expected number. Check your atomic structrure to ensure that it is realistic.

Too many atoms for dump dcd

>   The system size must fit in a 32-bit integer to use this dump style.

Too many atoms for dump xtc

>   The system size must fit in a 32-bit integer to use this dump style.

Too many atoms to dump sort

>   Cannot sort when running with more than 2^31 atoms.

Too many exponent bits for lookup table

>   Table size specified via pair_modify command does not work with your machine's floating point representation.

Too many groups

>   The maximum number of atom groups (including the "all" group) is given by MAX_GROUP in group.cpp and is 32.

Too many iterations

>   You must use a number of iterations that fit in a 32-bit integer for minimization.

Too many lines in one body in data file - boost MAXBODY

>   MAXBODY is a setting at the top of the src/read_data.cpp file. Set it larger and re-compile the code.

Too many local+ghost atoms for neighbor list

>   The number of nlocal + nghost atoms on a processor is limited by the size of a 32-bit integer with 2 bits removed for masking 1-2, 1-3, 1-4 neighbors.

Too many mantissa bits for lookup table

>   Table size specified via pair_modify command does not work with your machine's floating point representation.

Too many masses for fix shake

>   The fix shake command cannot list more masses than there are atom types.

Too many neighbor bins

>   This is likely due to an immense simulation box that has blown up to a large size.

Too many timesteps

>   The cummulative timesteps must fit in a 64-bit integer.

Too many timesteps for NEB

>   You must use a number of timesteps that fit in a 32-bit integer for NEB.

Too many total atoms

>   See the setting for bigint in the src/lmptype.h file.

Too many total bits for bitmapped lookup table

>   Table size specified via pair_modify command is too large. Note that a value of N generates a 2^N size table.

Too many touching neighbors - boost MAXTOUCH

A granular simulation has too many neighbors touching one atom. The MAXTOUCH parameter in fix_shear_history.cpp must be set larger and LIGGGHTS(R)-WSU must be re-built.

Too much per-proc info for dump

Number of local atoms times number of columns must fit in a 32-bit integer for dump.

Tree structure in joint connections

Fix poems cannot (yet) work with coupled bodies whose joints connect the bodies in a tree structure.

Triclinic box skew is too large

The displacement in a skewed direction must be less than half the box length in that dimension. E.g. the xy tilt must be between -half and +half of the x box length. This constraint can be relaxed by using the box tilt command.

Tried to convert a double to int, but input_double > INT_MAX

Self-explanatory.

Two groups cannot be the same in fix spring couple

Self-explanatory.

USER-CUDA mode requires CUDA variant of min style

CUDA mode is enabled, so the min style must include a cuda suffix.

USER-CUDA mode requires CUDA variant of run style

CUDA mode is enabled, so the run style must include a cuda suffix.

USER-CUDA package requires a cuda enabled atom_style

Self-explanatory.

Unable to initialize accelerator for use

There was a problem initializing an accelerator for the gpu package

Unbalanced quotes in input line

No matching end double quote was found following a leading double quote.

Unexpected end of -reorder file

Self-explanatory.

Unexpected end of custom file

Self-explanatory.

Unexpected end of data file

LIGGGHTS(R)-WSU hit the end of the data file while attempting to read a section. Something is wrong with the format of the data file.

Unexpected end of dump file

A read operation from the file failed.

Unexpected end of fix rigid file

A read operation from the file failed.

Units command after simulation box is defined

The units command cannot be used after a read_data, read_restart, or create_box command.

Universe/uloop variable count < # of partitions

A universe or uloop style variable must specify a number of values >= to the number of processor partitions.

Unknown command: %s

The command is not known to LIGGGHTS(R)-WSU. Check the input script.

Unknown error in GPU library

Self-explanatory.

Unknown identifier in data file: %s

A section of the data file cannot be read by LIGGGHTS(R)-WSU.

Unknown table style in angle style table

Self-explanatory.

Unknown table style in bond style table

Self-explanatory.

Unknown table style in pair_style command

Style of table is invalid for use with pair_style table command.

Unknown unit_style

Self-explanatory. Check the input script or data file.

Unrecognized lattice type in MEAM file 1

The lattice type in an entry of the MEAM library file is not valid.

Unrecognized lattice type in MEAM file 2

The lattice type in an entry of the MEAM parameter file is not valid.

Unrecognized pair style in compute pair command

Self-explanatory.

Unrecognized virial argument in pair_style command

Only two options are supported: LAMMPSvirial and KIMvirial

Unsupported mixing rule in kspace_style ewald/disp

Only geometric mixing is supported.

Unsupported mixing rule in kspace_style pppm/disp for pair_style %s

Only geometric mixing is supported.

Unsupported order in kspace_style ewald/disp

Only 1/r^6 dispersion terms are supported.

Unsupported order in kspace_style pppm/disp pair_style %s

Only 1/r^6 dispersion terms are supported.

Use of change_box with undefined lattice

Must use lattice command with displace_box command if units option is set to lattice.

Use of compute temp/ramp with undefined lattice

Must use lattice command with compute temp/ramp command if units option is set to lattice.

Use of displace_atoms with undefined lattice

Must use lattice command with displace_atoms command if units option is set to lattice.

Use of fix append/atoms with undefined lattice

A lattice must be defined before using this fix.

Use of fix ave/spatial with undefined lattice

A lattice must be defined to use fix ave/spatial with units = lattice.

Use of fix deform with undefined lattice

A lattice must be defined to use fix deform with units = lattice.

Use of fix deposit with undefined lattice

Must use lattice command with compute fix deposit command if units option is set to lattice.

Use of fix dt/reset with undefined lattice

Must use lattice command with fix dt/reset command if units option is set to lattice.

Use of fix indent with undefined lattice

The lattice command must be used to define a lattice before using the fix indent command.

Use of fix move with undefined lattice

Must use lattice command with fix move command if units option is set to lattice.

Use of fix recenter with undefined lattice

Must use lattice command with fix recenter command if units option is set to lattice.

Use of fix wall with undefined lattice

Must use lattice command with fix wall command if units option is set to lattice.

Use of fix wall/piston with undefined lattice

A lattice must be defined before using this fix.

Use of region with undefined lattice

If units = lattice (the default) for the region command, then a lattice must first be defined via the lattice command.

Use of velocity with undefined lattice

If units = lattice (the default) for the velocity set or velocity ramp command, then a lattice must first be defined via the lattice command.

Using fix nvt/sllod with inconsistent fix deform remap option

Fix nvt/sllod requires that deforming atoms have a velocity profile provided by "remap v" as a fix deform option.

Using fix nvt/sllod with no fix deform defined

Self-explanatory.

Using fix srd with inconsistent fix deform remap option

When shearing the box in an SRD simulation, the remap v option for fix deform needs to be used.

Using pair lubricate with inconsistent fix deform remap option

Must use remap v option with fix deform with this pair style.

Using pair lubricate/poly with inconsistent fix deform remap option

If fix deform is used, the remap v option is required.

Variable ID in variable formula does not exist

Self-explanatory.

Variable evaluation before simulation box is defined

Cannot evaluate a compute or fix or atom-based value in a variable before the simulation has been setup.

Variable evaluation in fix wall gave bad value

The returned value for epsilon or sigma < 0.0.

Variable evaluation in region gave bad value

Variable returned a radius < 0.0.

Variable for compute ti is invalid style

Self-explanatory.

Variable for dump every is invalid style

Only equal-style variables can be used.

Variable for dump image center is invalid style

Must be an equal-style variable.

Variable for dump image persp is invalid style

Must be an equal-style variable.

Variable for dump image phi is invalid style

Must be an equal-style variable.

Variable for dump image theta is invalid style

Must be an equal-style variable.

Variable for dump image zoom is invalid style

Must be an equal-style variable.

Variable for fix adapt is invalid style

Only equal-style variables can be used.

Variable for fix addforce is invalid style

Self-explanatory.

Variable for fix aveforce is invalid style

Only equal-style variables can be used.

Variable for fix deform is invalid style

The variable must be an equal-style variable.

Variable for fix efield is invalid style

Only equal-style variables can be used.

Variable for fix gravity is invalid style

Only equal-style variables can be used.

Variable for fix heat is invalid style

Only equal-style or atom-style variables can be used.

Variable for fix indent is invalid style

Only equal-style variables can be used.

Variable for fix indent is not equal style

Only equal-style variables can be used.

Variable for fix langevin is invalid style

It must be an equal-style variable.

Variable for fix move is invalid style

Only equal-style variables can be used.

Variable for fix setforce is invalid style

Only equal-style variables can be used.

Variable for fix temp/berendsen is invalid style

Only equal-style variables can be used.
Variable for fix temp/rescale is invalid style
Only equal-style variables can be used.
Variable for fix wall is invalid style
Only equal-style variables can be used.
Variable for fix wall/reflect is invalid style
Only equal-style variables can be used.
Variable for fix wall/srd is invalid style
Only equal-style variables can be used.
Variable for group is invalid style
Only atom-style variables can be used.
Variable for region cylinder is invalid style
Only equal-style varaibles are allowed.
Variable for region is invalid style
Only equal-style variables can be used.
Variable for region is not equal style
Self-explanatory.
Variable for region sphere is invalid style
Only equal-style varaibles are allowed.
Variable for restart is invalid style
Only equal-style variables can be used.
Variable for thermo every is invalid style
Only equal-style variables can be used.
Variable for velocity set is invalid style
Only atom-style variables can be used.
Variable formula compute array is accessed out-of-range
Self-explanatory.
Variable formula compute vector is accessed out-of-range
Self-explanatory.
Variable formula fix array is accessed out-of-range
Self-explanatory.
Variable formula fix vector is accessed out-of-range
Self-explanatory.
Variable has circular dependency
A circular dependency is when variable "a" in used by variable "b" and variable "b" is also used by varaible "a". Circular dependencies with longer chains of dependence are also not allowed.
Variable name for compute atom/molecule does not exist
Self-explanatory.
Variable name for compute reduce does not exist
Self-explanatory.
Variable name for compute ti does not exist
Self-explanatory.
Variable name for dump every does not exist
Self-explanatory.
Variable name for dump image center does not exist
Self-explanatory.
Variable name for dump image persp does not exist
Self-explanatory.
Variable name for dump image phi does not exist
Self-explanatory.
Variable name for dump image theta does not exist
Self-explanatory.
Variable name for dump image zoom does not exist
Self-explanatory.

Variable name for fix adapt does not exist
>   Self-explanatory.

Variable name for fix addforce does not exist
>   Self-explanatory.

Variable name for fix ave/atom does not exist
>   Self-explanatory.

Variable name for fix ave/correlate does not exist
>   Self-explanatory.

Variable name for fix ave/histo does not exist
>   Self-explanatory.

Variable name for fix ave/spatial does not exist
>   Self-explanatory.

Variable name for fix ave/time does not exist
>   Self-explanatory.

Variable name for fix aveforce does not exist
>   Self-explanatory.

Variable name for fix deform does not exist
>   Self-explantory.

Variable name for fix efield does not exist
>   Self-explanatory.

Variable name for fix gravity does not exist
>   Self-explanatory.

Variable name for fix heat does not exist
>   Self-explanatory.

Variable name for fix indent does not exist
>   Self-explanatory.

Variable name for fix langevin does not exist
>   Self-explanatory.

Variable name for fix move does not exist
>   Self-explanatory.

Variable name for fix setforce does not exist
>   Self-explanatory.

Variable name for fix store/state does not exist
>   Self-explanatory.

Variable name for fix temp/berendsen does not exist
>   Self-explanatory.

Variable name for fix temp/rescale does not exist
>   Self-explanatory.

Variable name for fix wall does not exist
>   Self-explanatory.

Variable name for fix wall/reflect does not exist
>   Self-explanatory.

Variable name for fix wall/srd does not exist
>   Self-explanatory.

Variable name for group does not exist
>   Self-explanatory.

Variable name for region cylinder does not exist
>   Self-explanatory.

Variable name for region does not exist
>   Self-explanatory.

Variable name for region sphere does not exist
>   Self-explanatory.

Variable name for restart does not exist
>   Self-explanatory.

Variable name for thermo every does not exist
> Self-explanatory.

Variable name for velocity set does not exist
> Self-explanatory.

Variable name must be alphanumeric or underscore characters
> Self-explanatory.

Velocity command before simulation box is defined
> The velocity command cannot be used before a read_data, read_restart, or create_box command.

Velocity command with no atoms existing
> A velocity command has been used, but no atoms yet exist.

Velocity ramp in z for a 2d problem
> Self-explanatory.

Velocity temperature ID does not compute temperature
> The compute ID given to the velocity command must compute temperature.

Verlet/split requires 2 partitions
> See the -partition command-line switch.

Verlet/split requires Rspace partition layout be multiple of Kspace partition layout in each dim
> This is controlled by the processors command.

Verlet/split requires Rspace partition size be multiple of Kspace partition size
> This is so there is an equal number of Rspace processors for every Kspace processor.

Virial was not tallied on needed timestep
> You are using a thermo keyword that requires potentials to have tallied the virial, but they didn't on this timestep. See the variable doc page for ideas on how to make this work.

Wall defined twice in fix wall command
> Self-explanatory.

Wall defined twice in fix wall/reflect command
> Self-explanatory.

Wall defined twice in fix wall/srd command
> Self-explanatory.

Water H epsilon must be 0.0 for pair style lj/cut/tip4p/long
> This is because LIGGGHTS(R)-WSU does not compute the Lennard-Jones interactions with these particles for efficiency reasons.

Water H epsilon must be 0.0 for pair style lj/long/tip4p/long
> This is because LIGGGHTS(R)-WSU does not compute the Lennard-Jones interactions with these particles for efficiency reasons.

World variable count doesn't match # of partitions
> A world-style variable must specify a number of values equal to the number of processor partitions.

Write_restart command before simulation box is defined
> The write_restart command cannot be used before a read_data, read_restart, or create_box command.

Zero length rotation vector with displace_atoms
> Self-explanatory.

Zero length rotation vector with fix move
> Self-explanatory.

Zero-length lattice orient vector
> Self-explanatory.

Warnings:

Adjusting Coulombic cutoff for MSM, new cutoff = %g
> The adjust/cutoff command is turned on and the Coulombic cutoff has been adjusted to match the user-specified accuracy.

Atom with molecule ID = 0 included in compute molecule group
> The group used in a compute command that operates on moleclues includes atoms with no molecule ID. This is probably not what you want.

Bond/angle/dihedral extent > half of periodic box length

This is a restriction because LIGGGHTS(R)-WSU can be confused about which image of an atom in the bonded interaction is the correct one to use. "Extent" in this context means the maximum end-to-end length of the bond/angle/dihedral. LIGGGHTS(R)-WSU computes this by taking the maximum bond length, multiplying by the number of bonds in the interaction (e.g. 3 for a dihedral) and adding a small amount of stretch.

Both groups in compute group/group have a net charge; the Kspace boundary correction to energy will be non-zero

Self-explantory.

Broken bonds will not alter angles, dihedrals, or impropers

See the doc page for fix bond/break for more info on this restriction.

Building an occasional neighobr list when atoms may have moved too far

This can cause LIGGGHTS(R)-WSU to crash when the neighbor list is built. The solution is to check for building the regular neighbor lists more frequently.

Cannot include log terms without 1/r terms; setting flagHI to 1

Self-explanatory.

Cannot include log terms without 1/r terms; setting flagHI to 1.

Self-explanatory.

Charges are set, but coulombic solver is not used

The atom style supports charge, but this KSpace style does not include long-range Coulombics.

Compute cna/atom cutoff may be too large to find ghost atom neighbors

The neighbor cutoff used may not encompass enough ghost atoms to perform this operation correctly.

Computing temperature of portions of rigid bodies

The group defined by the temperature compute does not encompass all the atoms in one or more rigid bodies, so the change in degrees-of-freedom for the atoms in those partial rigid bodies will not be accounted for.

Created bonds will not create angles, dihedrals, or impropers

See the doc page for fix bond/create for more info on this restriction.

Dihedral problem: %d %ld %d %d %d %d

Conformation of the 4 listed dihedral atoms is extreme; you may want to check your simulation geometry.

Dump dcd/xtc timestamp may be wrong with fix dt/reset

If the fix changes the timestep, the dump dcd file will not reflect the change.

Ewald/disp Newton solver failed, using old method to estimate g_ewald

Self-explanatory.

FENE bond too long: %ld %d %d %g

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

FENE bond too long: %ld %g

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

Fix SRD walls overlap but fix srd overlap not set

You likely want to set this in your input script.

Fix bond/swap will ignore defined angles

See the doc page for fix bond/swap for more info on this restriction.

Fix evaporate may delete atom with non-zero molecule ID

This is probably an error, since you should not delete only one atom of a molecule.

Fix move does not update angular momentum

Atoms store this quantity, but fix move does not (yet) update it.

Fix move does not update quaternions

Atoms store this quantity, but fix move does not (yet) update it.

Fix recenter should come after all other integration fixes

Other fixes may change the position of the center-of-mass, so fix recenter should come last.

Fix shake with rRESPA computes invalid pressures

This is a known bug in LIGGGHTS(R)-WSU that has not yet been fixed. If you use SHAKE with rRESPA and perform a constant volume simulation (e.g. using fix npt) this only affects the output pressure, not the dynamics of the simulation. If you use SHAKE with rRESPA and perform a constant pressure simulation (e.g. using fix npt) then you will be equilibrating to the wrong volume.

Fix srd SRD moves may trigger frequent reneighboring

This is because the SRD particles may move long distances.

Fix srd grid size > 1/4 of big particle diameter

This may cause accuracy problems.

Fix srd particle moved outside valid domain

This may indicate a problem with your simulation parameters.

Fix srd particles may move > big particle diameter

This may cause accuracy problems.

Fix srd viscosity < 0.0 due to low SRD density

This may cause accuracy problems.

Fix thermal/conductivity comes before fix ave/spatial

The order of these 2 fixes in your input script is such that fix thermal/conductivity comes first. If you are using fix ave/spatial to measure the temperature profile induced by fix viscosity, then this may cause a glitch in the profile since you are averaging immediately after swaps have occurred. Flipping the order of the 2 fixes typically helps.

Fix viscosity comes before fix ave/spatial

The order of these 2 fixes in your input script is such that fix viscosity comes first. If you are using fix ave/spatial to measure the velocity profile induced by fix viscosity, then this may cause a glitch in the profile since you are averaging immediately after swaps have occurred. Flipping the order of the 2 fixes typically helps.

For better accuracy use 'pair_modify table 0'

The user-specified force accuracy cannot be achieved unless the table feature is disabled by using 'pair_modify table 0'.

Geometric mixing assumed for 1/r^6 coefficients

Self-explanatory.

Group for fix_modify temp != fix group

The fix_modify command is specifying a temperature computation that computes a temperature on a different group of atoms than the fix itself operates on. This is probably not what you want to do.

Improper problem: %d %ld %d %d %d %d

Conformation of the 4 listed improper atoms is extreme; you may want to check your simulation geometry.

Inconsistent image flags

The image flags for a pair on bonded atoms appear to be inconsistent. Inconsistent means that when the coordinates of the two atoms are unwrapped using the image flags, the two atoms are far apart. Specifically they are further apart than half a periodic box length. Or they are more than a box length apart in a non-periodic dimension. This is usually due to the initial data file not having correct image flags for the 2 atoms in a bond that straddles a periodic boundary. They should be different by 1 in that case. This is a warning because inconsistent image flags will not cause problems for dynamics or most LIGGGHTS(R)-WSU simulations. However they can cause problems when such atoms are used with the fix rigid or replicate commands.

KIM Model does not provide `energy'; Potential energy will be zero

Self-explanatory.

KIM Model does not provide `forces'; Forces will be zero

Self-explanatory.

KIM Model does not provide `particleEnergy'; energy per atom will be zero

Self-explanatory.

KIM Model does not provide `particleVirial'; virial per atom will be zero

Self-explanatory.

Kspace_modify slab param < 2.0 may cause unphysical behavior

The kspace_modify slab parameter should be larger to insure periodic grids padded with empty space

do not overlap.

Less insertions than requested

Less atom insertions occurred on this timestep due to the fix pour command than were scheduled. This is probably because there were too many overlaps detected.

Library error in lammps_gather_atoms

This library function cannot be used if atom IDs are not defined or are not consecutively numbered.

Library error in lammps_scatter_atoms

This library function cannot be used if atom IDs are not defined or are not consecutively numbered, or if no atom map is defined. See the atom_modify command for details about atom maps.

Lost atoms via change_box: original %ld current %ld

The command options you have used caused atoms to be lost.

Lost atoms via displace_atoms: original %ld current %ld

The command options you have used caused atoms to be lost.

Lost atoms: original %ld current %ld

Lost atoms are checked for each time thermo output is done. See the thermo_modify lost command for options. Lost atoms usually indicate bad dynamics, e.g. atoms have been blown far out of the simulation box, or moved futher than one processor's sub-domain away before reneighboring.

MSM mesh too small, increasing to 2 points in each direction

Self-explanatory.

Mismatch between velocity and compute groups

The temperature computation used by the velocity command will not be on the same group of atoms that velocities are being set for.

Mixing forced for LJ coefficients

Self-explanatory.

Mixing forced for lj coefficients

Self-explanatory.

More than one compute centro/atom

It is not efficient to use compute centro/atom more than once.

More than one compute cluster/atom

It is not efficient to use compute cluster/atom more than once.

More than one compute cna/atom defined

It is not efficient to use compute cna/atom more than once.

More than one compute contact/atom

It is not efficient to use compute contact/atom more than once.

More than one compute coord/atom

It is not efficient to use compute coord/atom more than once.

More than one compute damage/atom

It is not efficient to use compute ke/atom more than once.

More than one compute erotate/sphere/atom

It is not efficient to use compute erorate/sphere/atom more than once.

More than one compute ke/atom

It is not efficient to use compute ke/atom more than once.

More than one compute voronoi/atom command

It is not efficient to use compute voronoi/atom more than once.

More than one fix poems

It is not efficient to use fix poems more than once.

More than one fix rigid

It is not efficient to use fix rigid more than once.

Neighbor exclusions used with KSpace solver may give inconsistent Coulombic energies

This is because excluding specific pair interactions also excludes them from long-range interactions which may not be the desired effect. The special_bonds command handles this consistently by insuring excluded (or weighted) 1-2, 1-3, 1-4 interactions are treated consistently by both the short-range pair style and the long-range solver. This is not done for exclusions of charged atom pairs via the neigh_modify exclude command.

New thermo_style command, previous thermo_modify settings will be lost
> If a thermo_style command is used after a thermo_modify command, the settings changed by the thermo_modify command will be reset to their default values. This is because the thermo_modify commmand acts on the currently defined thermo style, and a thermo_style command creates a new style.

No Kspace calculation with verlet/split
> The 2nd partition performs a kspace calculation so the kspace_style command must be used.

No fixes defined, atoms won't move
> If you are not using a fix like nve, nvt, npt then atom velocities and coordinates will not be updated during timestepping.

No joints between rigid bodies, use fix rigid instead
> The bodies defined by fix poems are not connected by joints. POEMS will integrate the body motion, but it would be more efficient to use fix rigid.

Not using real units with pair reax
> This is most likely an error, unless you have created your own ReaxFF parameter file in a different set of units.

Number of MSM mesh points increased to be a multiple of 2
> MSM requires that the number of grid points in each direction be a multiple of two and the number of grid points in one or more directions have been adjusted to meet this requirement.

OMP_NUM_THREADS environment is not set.
> This environment variable must be set appropriately to use the USER-OMP pacakge.

One or more atoms are time integrated more than once
> This is probably an error since you typically do not want to advance the positions or velocities of an atom more than once per timestep.

One or more compute molecules has atoms not in group
> The group used in a compute command that operates on moleclues does not include all the atoms in some molecules. This is probably not what you want.

One or more respa levels compute no forces
> This is computationally inefficient.

Pair COMB charge %.10f with force %.10f hit max barrier
> Something is possibly wrong with your model.

Pair COMB charge %.10f with force %.10f hit min barrier
> Something is possibly wrong with your model.

Pair brownian needs newton pair on for momentum conservation
> Self-explanatory.

Pair dpd needs newton pair on for momentum conservation
> Self-explanatory.

Pair dsmc: num_of_collisions > number_of_A
> Collision model in DSMC is breaking down.

Pair dsmc: num_of_collisions > number_of_B
> Collision model in DSMC is breaking down.

Particle deposition was unsuccessful
> The fix deposit command was not able to insert as many atoms as needed. The requested volume fraction may be too high, or other atoms may be in the insertion region.

Reducing PPPM order b/c stencil extends beyond nearest neighbor processor
> This may lead to a larger grid than desired. See the kspace_modify overlap command to prevent changing of the PPPM order.

Reducing PPPMDisp Coulomb order b/c stencil extends beyond neighbor processor.
> This may lead to a larger grid than desired. See the kspace_modify overlap command to prevent changing of the PPPM order.

Reducing PPPMDisp Dispersion order b/c stencil extends beyond neighbor processor
> This may lead to a larger grid than desired. See the kspace_modify overlap command to prevent changing of the PPPM order.

Replacing a fix, but new group != old group

The ID and style of a fix match for a fix you are changing with a fix command, but the new group you are specifying does not match the old group.

Replicating in a non-periodic dimension

The parameters for a replicate command will cause a non-periodic dimension to be replicated; this may cause unwanted behavior.

Resetting reneighboring criteria during PRD

A PRD simulation requires that neigh_modify settings be delay = 0, every = 1, check = yes. Since these settings were not in place, LIGGGHTS(R)-WSU changed them and will restore them to their original values after the PRD simulation.

Resetting reneighboring criteria during TAD

A TAD simulation requires that neigh_modify settings be delay = 0, every = 1, check = yes. Since these settings were not in place, LIGGGHTS(R)-WSU changed them and will restore them to their original values after the PRD simulation.

Resetting reneighboring criteria during minimization

Minimization requires that neigh_modify settings be delay = 0, every = 1, check = yes. Since these settings were not in place, LIGGGHTS(R)-WSU changed them and will restore them to their original values after the minimization.

Restart file used different # of processors

The restart file was written out by a LIGGGHTS(R)-WSU 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 LIGGGHTS(R)-WSU simulation running on a different 3d grid of processors. Due to round-off, the trajectories of your restarted simulation may diverge a little more quickly than if you ran on the same # of processors.

Restart file used different boundary settings, using restart file values

Your input script cannot change these restart file settings.

Restart file used different newton bond setting, using restart file value

The restart file value will override the setting in the input script.

Restart file used different newton pair setting, using input script value

The input script value will override the setting in the restart file.

Restart file version does not match LIGGGHTS(R)-WSU version

This may cause problems when reading the restart file.

Restrain problem: %d %ld %d %d %d %d

Conformation of the 4 listed dihedral atoms is extreme; you may want to check your simulation geometry.

Running PRD with only one replica

This is allowed, but you will get no parallel speed-up.

SRD bin shifting turned on due to small lamda

This is done to try to preserve accuracy.

SRD bin size for fix srd differs from user request

Fix SRD had to adjust the bin size to fit the simulation box. See the cubic keyword if you want this message to be an error vs warning.

SRD bins for fix srd are not cubic enough

The bin shape is not within tolerance of cubic. See the cubic keyword if you want this message to be an error vs warning.

SRD particle %d started inside big particle %d on step %ld bounce %d

See the inside keyword if you want this message to be an error vs warning.

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. LIGGGHTS(R)-WSU will set it to 0.0 and continue.

Should not allow rigid bodies to bounce off relecting walls

LIGGGHTS(R)-WSU allows this, but their dynamics are not computed correctly.

System is not charge neutral, net charge = %g

Warnings: 524

The total charge on all atoms on the system is not 0.0, which is not valid for the long-range Coulombic solvers.

Table inner cutoff >= outer cutoff

You specified an inner cutoff for a Coulombic table that is longer than the global cutoff. Probably not what you wanted.

Temperature for MSST is not for group all

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

Temperature for NPT is not for group all

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

Temperature for fix modify is not for group all

The temperature compute is being used with a pressure calculation which does operate on group all, so this may be inconsistent.

Temperature for thermo pressure is not for group all

User-assigned temperature to thermo via the thermo_modify command does not compute temperature for all atoms. Since thermo computes a global pressure, the kinetic energy contribution from the temperature is assumed to also be for all atoms. Thus the pressure printed by thermo could be inaccurate.

Too many common neighbors in CNA %d times

More than the maximum # of neighbors was found multiple times. This was unexpected.

Too many inner timesteps in fix ttm

Self-explanatory.

Too many neighbors in CNA for %d atoms

More than the maximum # of neighbors was found multiple times. This was unexpected.

Triclinic box skew is large

The displacement in a skewed direction is normally required to be less than half the box length in that dimension. E.g. the xy tilt must be between -half and +half of the x box length. You have relaxed the constraint using the box tilt command, but the warning means that a LIGGGHTS(R)-WSU simulation may be inefficient as a result.

Use special bonds = 0,1,1 with bond style fene

Most FENE models need this setting for the special_bonds command.

Use special bonds = 0,1,1 with bond style fene/expand

Most FENE models need this setting for the special_bonds command.

Using compute temp/deform with inconsistent fix deform remap option

Fix nvt/sllod assumes deforming atoms have a velocity profile provided by "remap v" or "remap none" as a fix deform option.

Using compute temp/deform with no fix deform defined

This is probably an error, since it makes little sense to use compute temp/deform in this case.

Using fix srd with box deformation but no SRD thermostat

The deformation will heat the SRD particles so this can be dangerous.

Using largest cutoff for buck/long/coul/long

Self-exlanatory.

Using largest cutoff for pair_style lj/long/coul/long

Self-explanatory.

Using largest cutoff for pair_style lj/long/tip4p/long

Self-explanatory.

Using pair tail corrections with nonperiodic system

This is probably a bogus thing to do, since tail corrections are computed by integrating the density of a periodic system out to infinity.

5. Contact models

This section describes what granular models can be used along with pair gran and fix wall/gran

cohesion commands

Click on the style itself for a full description:

| easo/capillary/viscous | sjkr | sjkr2 | washino/capillary/viscous |

model commands

Click on the style itself for a full description:

| hertz | hertz/stiffness | hooke | hooke/stiffness |

rolling_friction commands

Click on the style itself for a full description:

| cdt | epsd | epsd2 | epsd3 |

surface commands

Click on the style itself for a full description:

| sphere |

tangential commands

Click on the style itself for a full description:

| history | no_history |

5. Contact models

527

This section describes what granular models can be used along with pair gran and fix wall/gran

THIS IS PARSING LINE

7. How-to discussions

This section describes how to perform common tasks using LIGGGHTS(R)-WSU.

The example input scripts included in the LIGGGHTS(R)-WSU distribution and highlighted in Section_example also show how to setup and run various kinds of simulations.

7.1 Restarting a simulation

There are 3 ways to continue a long LIGGGHTS(R)-WSU 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 using the -r command-line switch and read by a read_data command in a new script.

The read_restart command typically replaces the "create_box"create_box.html command in the input script that restarts the simulation.

Note that while the internal state of the simulation is stored in the restart file, most material properties and simulation settings are not stored in the restart file. Be careful when changing settings and/or material parameters when restarting a simulation.

The following commands do not need to be repeated because their settings are included in the restart file: *units, atom_style*.

As an alternate approach, the restart file could be converted to a data file as follows:

```
lmp_g++ -r tmp.restart.50 tmp.restart.data
```

Then, the "read_data"read_data.html command can be used to restart the simulation.

reset_timestep command can be used to tell LIGGGHTS(R)-WSU the current timestep.

7.2 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 LIGGGHTS(R)-WSU distribution are for 2d models.

IMPORTANT NOTE: Some models in LIGGGHTS(R)-WSU treat particles as finite-size spheres, as opposed to point particles. In 2d, the particles will still be spheres, not disks, meaning their moment of inertia will be the same as in 3d.

7.3 Running multiple simulations from one input script

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

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

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

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

If you wish to run totally different simulations, one after the other, the [clear](#) command can be used in between them to re-initialize LIGGGHTS(R)-WSU. For example, this script

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

would run 2 independent simulations, one after the other.

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

in.polymer

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

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

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

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

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

7.4 Granular models

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

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

- atom_style sphere
- fix nve/sphere
- fix gravity

This compute

- compute erotate/sphere

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

Use a number of particle-particle and particle-wall contact models, which compute forces and torques between interacting pairs of particles and between particles and primitive walls or mesh elements. Details can be found here:

- pair_style gran
- List of all contact models

These commands implement fix options specific to granular systems:

- fix freeze
- fix pour
- fix viscous
- fix wall/gran

fix wall/gran commands can define two types of walls: primitive and mesh. Mesh walls are defined using a fix mesh/surface or related command.

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

7.5 Coupling LIGGGHTS(R)-WSU to other codes

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

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

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

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

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

See Section_modify of the documentation for how to add a new command to LIGGGHTS(R)-WSU.

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

Examples of driver codes that call LIGGGHTS(R)-WSU as a library are included in the examples/COUPLE directory of the LIGGGHTS(R)-WSU distribution; see examples/COUPLE/README for more details:

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

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

The files src/library.cpp and library.h contain the C-style interface to LIGGGHTS(R)-WSU. See Section_howto 19 of the manual for a description of the interface and how to extend it for your needs.

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

7.6 Visualizing LIGGGHTS(R)-WSU snapshots

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

LIGGGHTS(R)-WSU snapshots are created by the dump command which can create files in several formats. The native LIGGGHTS(R)-WSU dump format is a text file (see "dump atom" or "dump custom").

A Python-based toolkit (LPP) distributed by our group can read native LIGGGHTS(R)-WSU dump files, including custom dump files with additional columns of user-specified atom information, and convert them to VTK file formats that can be read with Paraview.

Also, there exist a number of dump commands that output to VTK natively.

7.7 Triclinic (non-orthogonal) simulation boxes

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

LIGGGHTS(R)-WSU also allows simulations to be performed in triclinic (non-orthogonal) simulation boxes shaped as a parallelepiped with triclinic symmetry. The parallelepiped has its "origin" at (xlo,ylo,zlo) and is defined by 3 edge vectors starting from the origin given by $\mathbf{a}$ = (xhi-xlo,0,0); $\mathbf{b}$ = (xy,yhi-ylo,0); $\mathbf{c}$ = (xz,yz,zhi-zlo). *xy,xz,yz* can be 0.0 or positive or negative values and are called "tilt factors" because they are the amount of displacement applied to faces of an originally orthogonal box to transform it into the parallelepiped. In LIGGGHTS(R)-WSU the triclinic simulation box edge vectors $\mathbf{a}$, $\mathbf{b}$, and $\mathbf{c}$ cannot be arbitrary vectors. As indicated, $\mathbf{a}$ must lie on the positive x axis. $\mathbf{b}$ must lie in the xy plane, with strictly positive y component. $\mathbf{c}$ may have any orientation with strictly positive z component. The requirement that $\mathbf{a}$, $\mathbf{b}$, and $\mathbf{c}$ have strictly positive x, y, and z components, respectively, ensures that $\mathbf{a}$, $\mathbf{b}$, and $\mathbf{c}$ form a complete right-handed basis. These restrictions impose no loss of generality, since it is possible to rotate/invert any set of 3 crystal basis vectors so that they conform to the restrictions.

For example, assume that the 3 vectors $\mathbf{A}$,$\mathbf{B}$,$\mathbf{C}$ are the edge vectors of a general parallelepiped, where there is no restriction on $\mathbf{A}$,$\mathbf{B}$,$\mathbf{C}$ other than they form a complete right-handed basis i.e. $\mathbf{A}$ x $\mathbf{B}$. $\mathbf{C}$ > 0. The equivalent LIGGGHTS(R)-WSU $\mathbf{a}$,$\mathbf{b}$,$\mathbf{c}$ are a linear rotation of $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ and can be computed as follows:

$$(\mathbf{a} \quad \mathbf{b} \quad \mathbf{c}) = \begin{pmatrix} a_x & b_x & c_x \\ 0 & b_y & c_y \\ 0 & 0 & c_z \end{pmatrix}$$

$$a_x = A$$

$$b_x = \mathbf{B} \cdot \hat{\mathbf{A}} = B\cos\gamma$$

$$b_y = |\hat{\mathbf{A}} \times \mathbf{B}| = B\sin\gamma = \sqrt{B^2 - b_x^2}$$

$$c_x = \mathbf{C} \cdot \hat{\mathbf{A}} = C\cos\beta$$

$$c_y = \mathbf{C} \cdot \widehat{(\mathbf{A} \times \mathbf{B})} \times \hat{\mathbf{A}} = \frac{\mathbf{B} \cdot \mathbf{C} - b_x c_x}{b_y}$$

$$c_z = |\mathbf{C} \cdot \widehat{(\mathbf{A} \times \mathbf{B})}| = \sqrt{C^2 - c_x^2 - c_y^2}$$

where A = |**A**| indicates the scalar length of **A**. The ^ hat symbol indicates the corresponding unit vector. *beta* and *gamma* are angles between the vectors described below. Note that by construction, **a**, **b**, and **c** have strictly positive x, y, and z components, respectively. If it should happen that **A**, **B**, and **C** form a left-handed basis, then the above equations are not valid for **c**. In this case, it is necessary to first apply an inversion. This can be achieved by interchanging two basis vectors or by changing the sign of one of them.

For consistency, the same rotation/inversion applied to the basis vectors must also be applied to atom positions, velocities, and any other vector quantities. This can be conveniently achieved by first converting to fractional coordinates in the old basis and then converting to distance coordinates in the new basis. The transformation is given by the following equation:

$$
\mathbf{x} = \begin{pmatrix} \mathbf{a} & \mathbf{b} & \mathbf{c} \end{pmatrix} \cdot \frac{1}{V} \begin{pmatrix} \mathbf{B} \times \mathbf{C} \\ \mathbf{C} \times \mathbf{A} \\ \mathbf{A} \times \mathbf{B} \end{pmatrix} \cdot \mathbf{X}
$$

where *V* is the volume of the box, **X** is the original vector quantity and **x** is the vector in the LIGGGHTS(R)-WSU basis.

There is no requirement that a triclinic box be periodic in any dimension, though it typically should be in at least the 2nd dimension of the tilt (y in xy) if you want to enforce a shift in periodic boundary conditions across that boundary. Some commands that work with triclinic boxes, e.g. the fix deform and fix npt commands, require periodicity or non-shrink-wrap boundary conditions in specific dimensions. See the command doc pages for details.

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

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

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

To avoid extremely tilted boxes (which would be computationally inefficient), LIGGGHTS(R)-WSU normally requires that no tilt factor can skew the box more than half the distance of the parallel box length, which is the 1st dimension in the tilt factor (x for xz). This is required both when the simulation box is created, e.g. via the create_box or read_data commands, as well as when the box shape changes dynamically during a simulation, e.g. via the fix deform command.

For example, if xlo = 2 and xhi = 12, then the x box length is 10 and the xy tilt factor must be between -5 and 5. Similarly, both xz and yz must be between -(xhi-xlo)/2 and +(yhi-ylo)/2. Note that this is not a limitation, since if the maximum tilt factor is 5 (as in this example), then configurations with tilt = ..., -15, -5, 5, 15, 25,

... are geometrically all equivalent. If the box tilt exceeds this limit during a dynamics run (e.g. via the fix deform command), then the box is "flipped" to an equivalent shape with a tilt factor within the bounds, so the run can continue. See the fix deform doc page for further details.

One exception to this rule is if the 1st dimension in the tilt factor (x for xy) is non-periodic. In that case, the limits on the tilt factor are not enforced, since flipping the box in that dimension does not change the atom positions due to non-periodicity. In this mode, if you tilt the system to extreme angles, the simulation will simply become inefficient, due to the highly skewed simulation box.

The limitation on not creating a simulation box with a tilt factor skewing the box more than half the distance of the parallel box length can be overridden via the box command. Setting the *tilt* keyword to *large* allows any tilt factors to be specified.

Note that if a simulation box has a large tilt factor, LIGGGHTS(R)-WSU will run less efficiently, due to the large volume of communication needed to acquire ghost atoms around a processor's irregular-shaped sub-domain. For extreme values of tilt, LIGGGHTS(R)-WSU may also lose atoms and generate an error.

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

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

The inverse relationship can be written as follows:

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

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

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

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

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

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

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

7.8 Output from LIGGGHTS(R)-WSU (thermo, dumps, computes, fixes, variables)

There are four basic kinds of LIGGGHTS(R)-WSU output:

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

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

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

The following sub-sections discuss different LIGGGHTS(R)-WSU command related to output and the kind of data they operate on and produce:

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

Global/per-atom/local data

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

Scalar/vector/array data

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

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

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

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

Thermodynamic output

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

Note that thermodynamic output values can be "extensive" or "intensive". The former scale with the number of atoms in the system (e.g. total energy), the latter do not (e.g. temperature). The setting for thermo_modify norm determines whether extensive quantities are normalized or not. Computes and fixes produce either extensive or intensive values; see their individual doc pages for details. Equal-style variables produce only intensive values; you can include a division by "natoms" in the formula if desired, to make an extensive calculation produce an intensive result.

Dump file output

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

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

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

Fixes that write output files

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

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

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

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

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

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

Computes that process output quantities

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

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

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

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

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

Fixes that process output quantities

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

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

Computes that generate values to output

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

Fixes that generate values to output

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

Variables that generate values to output

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

Computes that process output quantities 539

described in this section.

Summary table of output options and data flow between commands

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

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

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

7.9 Walls

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

Walls in LIGGGHTS(R)-WSU for granular simulations are typically defined using fix wall/gran. This command can define two types of walls: primitive and mesh. Mesh walls are defined using a fix mesh/surface or related command.

Also. walls can be constructed made of particles

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

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

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

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

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

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

- fix wall/reflect - reflective flat walls
- fix wall/region - use region surface as wall

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

7.10 Library interface to LIGGGHTS(R)-WSU

As described in Section_start 5, LIGGGHTS(R)-WSU can be built as a library, so that it can be called by another code, used in a coupled manner with other codes, or driven through a Python interface.

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

Library.cpp contains these 4 functions:

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

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

LIGGGHTS(R)-WSU will run on the set of processors in the communicator. This means the calling code can run LIGGGHTS(R)-WSU on all or a subset of processors. For example, a wrapper script might decide to alternate between LIGGGHTS(R)-WSU and another code, allowing them both to run on all the processors. Or it might allocate half the processors to LIGGGHTS(R)-WSU and half to the other code and run both codes simultaneously before syncing them up periodically. Or it might instantiate multiple instances of LIGGGHTS(R)-WSU to perform different calculations.

The lammps_close() function is used to shut down an instance of LIGGGHTS(R)-WSU and free all its memory.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

3. Commands

This section describes how a LIGGGHTS(R)-WSU input script is formatted and the input script commands used to define a LIGGGHTS(R)-WSU simulation.

3.1 LIGGGHTS(R)-WSU input script
3.2 Parsing rules
3.3 Input script structure
3.4 An example input script

3.1 LIGGGHTS(R)-WSU input script

LIGGGHTS(R)-WSU executes by reading commands from a input script (text file), one line at a time. When the input script ends, LIGGGHTS(R)-WSU exits. Each command causes LIGGGHTS(R)-WSU 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) LIGGGHTS(R)-WSU 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 LIGGGHTS(R)-WSU how to map processors to the simulation box.

Many input script errors are detected by LIGGGHTS(R)-WSU 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. LIGGGHTS(R)-WSU commands are case sensitive. Command names are lower-case, as are specified command arguments. Upper case letters may be used in file names or user-chosen ID strings.

Here is how each line in the input script is parsed by LIGGGHTS(R)-WSU:

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

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

(3) The line is searched repeatedly for $ characters, which indicate variables that are replaced with a text string. See an exception in (6).

If the $ is followed by curly brackets, then the variable name is the text inside the curly brackets. If no curly brackets follow the $, then the variable name is the single character immediately following the $. Thus ${myTemp} and $x refer to variable names "myTemp" and "x".

If the $ is followed by parenthesis, then the text inside the parenthesis is treated as an "immediate" variable and evaluated as an [equal-style variable](). This is a way to use numeric formulas in an input script without having to assign them to variable names. For example, these 3 input script lines:

```
variable X equal (xlo+xhi)/2+sqrt(v_area)
region 1 block $X 2 INF INF EDGE EDGE
variable X delete
```

can be replaced by

```
region 1 block $((xlo+xhi)/2+sqrt(v_area)) 2 INF INF EDGE EDGE
```

so that you do not have to define (or discard) a temporary variable X.

Note that neither the curly-bracket or immediate form of variables can contain nested $ characters for other variables to substitute for. Thus you cannot do this:

```
variable        a equal 2
variable        b2 equal 4
print           "B2 = ${b$a}"
```

Nor can you specify this $($x-1.0) for an immediate variable, but you could use $(v_x-1.0), since the latter is valid syntax for an [equal-style variable]().

See the [variable]() command for more details of how strings are assigned to variables and evaluated, and how they can be used in input script commands.

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

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

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

```
print "Volume = $v"
print 'Volume = $v'
if "$steps > 1000" then quit
```

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

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

3.3 Input script structure

This section describes the structure of a typical LIGGGHTS(R)-WSU input script. The "examples" directory in the LIGGGHTS(R)-WSU distribution contains many sample input scripts; the corresponding problems are discussed in <u>Section_example</u>, and animated on the <u>LIGGGHTS(R)-WSU WWW Site</u>.

A LIGGGHTS(R)-WSU input script typically has 4 parts:

1. Initialization
2. Atom/particle definition and insertion
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/particles are created or read-in from a file.

The relevant commands are <u>units</u>, <u>dimension</u>, <u>newton</u>, <u>processors</u>, <u>boundary</u>, <u>atom_style</u>, <u>atom_modify</u>.

If force-field parameters appear in the files that will be read, these commands tell LIGGGHTS(R)-WSU what kinds of force fields are being used: <u>pair_style</u>, <u>bond_style</u>, <u>fix wall/gran</u> .

Granular walls typically require meshes to be used. See <u>fix mesh/surface</u> for details.

(2) Atom/particle definition and insertion

There are 3 ways to define atoms in LIGGGHTS(R)-WSU. Read them in from a data or restart file via the <u>read_data</u> or <u>read_restart</u> commands. Or create atoms on a lattice using these commands: <u>lattice</u>, <u>region</u>, <u>create_box</u>, <u>create_atoms</u>.

However, the most common way to insert granular particles is to use one of the fix insert/* commands: <u>fix insert/pack</u>, "fix insert/stream"stream.html, <u>fix insert/rate/region</u>

Before these insertion commands can be used, particle distributions (fix particledistribution/discrete) are built up using particle templates. For spherical particles, such particle templates are defined using fix particletemplate/sphere.

(3) Settings

Once atoms are defined, a variety of settings can be specified: simulation parameters, output options, etc.

Material parameters and force field coefficients are set by these commands fix property/global, or pair_coeff, bond_coeff.

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

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

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

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

(4) Run a simulation

A simulation is run using the run command.

3.4 An example input script

This shows an example input script for a LIGGGHTS(R)-WSU simulation.

```
#Contact model example
atom_style  granular
atom_modify  map array
boundary    m m m
newton      off
communicate     single vel yes
units       si
region      reg block -0.05 0.05 -0.05 0.05 0. 0.15 units box
create_box  1 reg
neighbor    0.002 bin
neigh_modify delay 0
#Material properties required for pair style
fix         m1 all property/global youngsModulus peratomtype 5.e6
fix         m2 all property/global poissonsRatio peratomtype 0.45
fix         m3 all property/global coefficientRestitution peratomtypepair 1 0.95
fix         m4 all property/global coefficientFriction peratomtypepair 1 0.05
pair_style  gran model hertz tangential history
pair_coeff      * *
timestep    0.00001
fix         gravi all gravity 9.81 vector 0.0 0.0 -1.0
fix         zwalls1 all wall/gran model hertz tangential history primitive type 1 zplane 0.0
fix         zwalls2 all wall/gran model hertz tangential history primitive type 1 zplane 0.15
fix         cylwalls all wall/gran model hertz tangential history primitive type 1 zcylinder 0.0
#region of insertion
region      bc cylinder z 0. 0. 0.045 0.00 0.15 units box
#particle distributions
fix         pts1 all particletemplate/sphere 1 atom_type 1 density constant 2500 radius constant
fix         pdd1 all particledistribution/discrete 1.  1 pts1 1.0
fix         ins all insert/pack seed 100001 distributiontemplate pdd1 vel constant 0. 0. -0.5 &
```

```
              insert_every once overlapcheck yes all_in yes particles_in_region 1800 region bc
#apply nve integration to all particles
fix             integr all nve/sphere
#output settings, include total thermal energy
compute         rke all erotate/sphere
thermo_style    custom step atoms ke c_rke vol
thermo          1000
thermo_modify   lost ignore norm no
compute_modify  thermo_temp dynamic yes
dump            dmp all custom 800 post/dump*.newModels id type x y z ix iy iz vx vy vz fx fy fz
#insert particles and run
run             5000
```

#apply nve integration to all particles

#output settings, include total thermal energy

1. Introduction

This section provides an overview of what LIGGGHTS(R)-WSU can do, describes what it means for LIGGGHTS(R)-WSU to be an open-source code, and acknowledges the funding and people who have contributed to LIGGGHTS(R)-WSU over the years.

1.1 What is LIGGGHTS(R)-WSU
1.2 LIGGGHTS(R)-WSU features
1.3 Open source distribution
1.4 Acknowledgments and citations

1.1 What is LIGGGHTS(R)-WSU

LIGGGHTS(R)-WSU is an Open Source Discrete Element Method Particle Simulation Software.

LIGGGHTS (R) stands for LAMMPS improved for general granular and granular heat transfer simulations. LAMMPS is a classical molecular dynamics simulator. It is widely used in the field of Molecular Dynamics. Thanks to physical and algorithmic analogies, LAMMPS offers basic functionalities for DEM simulations. LIGGGHTS (R) aims to improve those capability with the goal to apply it to industrial applications. LIGGGHTSÂ® is currently used by a variety of research institutions world-wide. A number of Blue Chip companies in the fields of chemical, consumer goods, pharmaceutical, agricultural engineering, food production, steel production, mining, plastics production use LIGGGHTS (R) for improvement of production processes. LIGGGHTS(R)-WSU 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. LIGGGHTS (R) can model systems with only a few particles up to millions or billions. LIGGGHTS (R) is designed to be easy to modify or extend with new capabilities, such as new force fields, atom types, boundary conditions, or diagnostics. See Section_modify for more details.

LIGGGHTS(R)-WSU 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.

LIGGGHTS-PUBLIC VS. LIGGGHTS-PREMIUM

There are 2 flavors of LIGGGHTS. This documentation may refer to LIGGGHTS as LIGGGHTS-PUBLIC or as LIGGGHTS-PREMIUM, where PREMIUM is the name of your company or institution

LIGGGHTS-PUBLIC is the version of LIGGGHTS which is available for public download at this cite. It offers everything a researcher needs to do simulations: A large model portfolio, performance, and it is easy to understand and extend with new capabilities. It is periodically updated, and includes all the contributions from the community. LIGGGHTS-PREMIUM is a version with additional features for large-scale industrial application, available for industrial partner companies of CFDEM(R)project from all around the world, who have spent several million Euros for developing LIGGGHTS(R)-WSU. The Premium version is made available to industrial partners and customers as part of a long-term collaboration involving development projects to further strengthen the model porfolio of CFDEM(R)project.

History

As the sanme implies, some parts of LIGGGHTS(R)-WSU are based on LAMMPS. LAMMPS is a classical molecular dynamics simulation code designed to run efficiently on parallel computers. It was developed at Sandia National Laboratories, a US Department of Energy facility, with funding from the DOE. It is an open-source code, distributed freely under the terms of the GNU Public License (GPL). The primary developers of LAMMPS are Steve Plimpton, Aidan Thompson, and Paul Crozier. The LAMMPS WWW Site at http://lammps.sandia.gov has more information about LAMMPS. LAMMPS was originally developed under a US Department of Energy CRADA (Cooperative Research and Development Agreement) between two DOE labs and 3 companies.

1.2 LIGGGHTS(R)-WSU features

In the most general sense, LIGGGHTS(R)-WSU 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 LIGGGHTS(R)-WSU 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, LIGGGHTS(R)-WSU 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.

LIGGGHTS(R)-WSU General features

- runs on a single processor or in parallel
- distributed-memory message-passing parallelism (MPI)
- spatial-decomposition of simulation domain for parallelism
- open-source distribution
- highly portable C++
- optional libraries used: MPI
- easy to extend with new features and functionality
- runs from an input script
- syntax for defining and using variables and formulas
- syntax for looping over runs and breaking out of loops
- run one or multiple simulations simultaneously (in parallel) from one script
- output to the widely used, open source VTK data format
- build as library, invoke LIGGGHTS(R)-WSU thru library interface or provided Python wrapper
- couple with other codes: LIGGGHTS(R)-WSU calls other code, other code calls LIGGGHTS(R)-WSU, umbrella code calls both
- a strong eco-system of fellow simulation engines for co-simulation, efficienty and tightly coupled via MPI.
- LIGGGHTS(R)-WSU can be coupled to CFDEM(R)coupling for CFD-DEM simulations and Lagrange-Euler coupling in general
- LIGGGHTS(R)-WSU can be coupled to the simulation engine ParScale for the modelling of intra-particle transport processes

LIGGGHTS(R)-WSU Model features: LIGGGHTS-PUBLIC features for

- import and handling of complex geometries: STL walls and VTK tet volume meshes
- moving mesh feature with a varierty of motion schemes and a model for conveyor belts
- force and wear analysis on meshes as well as stress-controlled walls
- a variety of particle-particle contact implementations, including models for tangential history, non-spericity and cohesion

- interface to easily extend contact implementations
- heat conduction between particles
- particle insertion based on pre-defined volumes, meshes and particle streams from faces as well as particle growth and shrinkage
- flexible definition of particle distributions
- smoothed Particle Hydrodynamics (SPH) fluid models

LIGGGHTS(R)-WSU Model features: Additional features for LIGGGHTS(R)-WSU

LIGGGHTS versions other than LIGGGHTS-PUBLIC have additional functionalities, which are all described in the doc pages for the different commands.

1.3 Open source distribution

LIGGGHTS(R)-WSU 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 LIGGGHTS(R)-WSU distribution.

Here is a summary of what the GPL means for LIGGGHTS(R)-WSU users:

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

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

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

(4) If you give LIGGGHTS(R)-WSU files to someone else, the GPL LICENSE file and source file headers (including the copyright and GPL notices) must remain part of the code.

In the spirit of an open-source code, these are various ways you can contribute to making LIGGGHTS(R)-WSU better. If you find an error, omission or bug in this manual or in the code, please see our website for more info on how to get involved

- If you publish a paper using LIGGGHTS(R)-WSU results, send the citation (and any cool pictures or movies if you like) to add to the Publications, Pictures, and Movies pages of the our website, with links and attributions back to you.

1.4 Acknowledgments and citations

LIGGGHTS(R)-WSU development has been funded by a variety of sources:

- A variety of industrial partners in bi-laterial projects
- Christian Doppler Forschungsgesellschaft, www.cdg.at
- The EU FP7 programmes NanoSim and T-MAPPP
- The Austrial funding agency FFG, www.ffg.at

As LIGGGHTS(R)-WSU is based on LAMMPS, we also acknowledge the funding that helped creating LAMMPS. LAMMPS development has been funded by the US Department of Energy (DOE), through its

The following paper describe the basic parallel algorithms used in LIGGGHTS(R)-WSU. If you use LIGGGHTS(R)-WSU results in your published work, please cite this paper and include a pointer to the http://www.cfdem.com site

8. Modifying & extending LIGGGHTS(R)-WSU

This section describes how to customize LIGGGHTS(R)-WSU by modifying and extending its source code.

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

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

The best way to add a new feature is to find a similar feature in LIGGGHTS(R)-WSU 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 LIGGGHTS(R)-WSU and its class organization, but functions (class methods) that do actual computations are written in vanilla C-style code and operate on simple C-style data structures (vectors and arrays).

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

The advantage of C++ and its object-orientation is that all the code and variables needed to define the new feature are in the 2 files you write, and thus shouldn't make the rest of LIGGGHTS(R)-WSU 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 LIGGGHTS(R)-WSU input script with a command like

```
pair_style foo 0.1 3.5
```

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

```
#ifdef PAIR_CLASS
```

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

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

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

As illustrated by this pairwise example, many kinds of options are referred to in the LIGGGHTS(R)-WSU documentation as the "style" of a particular command.

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

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

Here are additional guidelines for modifying LIGGGHTS(R)-WSU 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 LIGGGHTS(R)-WSU performance when it isn't used, send an email to the **developers**. We might be interested in adding it to the LIGGGHTS(R)-WSU distribution. See further details on this at the bottom of this page.

8.1 Atom styles

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

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

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

init	one time setup (optional)
grow	re-allocate atom arrays to longer lengths (required)
grow_reset	make array pointers in Atom and AtomVec classes consistent (required)

copy	copy info for one atom to another atom's array locations (required)
pack_comm	store an atom's info in a buffer communicated every timestep (required)
pack_comm_vel	add velocity info to communication buffer (required)
pack_comm_hybrid	store extra info unique to this atom style (optional)
unpack_comm	retrieve an atom's info from the buffer (required)
unpack_comm_vel	also retrieve velocity info (required)
unpack_comm_hybrid	retreive extra info unique to this atom style (optional)
pack_reverse	store an atom's info in a buffer communicating partial forces (required)
pack_reverse_hybrid	store extra info unique to this atom style (optional)
unpack_reverse	retrieve an atom's info from the buffer (required)
unpack_reverse_hybrid	retreive extra info unique to this atom style (optional)
pack_border	store an atom's info in a buffer communicated on neighbor re-builds (required)
pack_border_vel	add velocity info to buffer (required)
pack_border_hybrid	store extra info unique to this atom style (optional)
unpack_border	retrieve an atom's info from the buffer (required)
unpack_border_vel	also retrieve velocity info (required)
unpack_border_hybrid	retreive extra info unique to this atom style (optional)
pack_exchange	store all an atom's info to migrate to another processor (required)
unpack_exchange	retrieve an atom's info from the buffer (required)
size_restart	number of restart quantities associated with proc's atoms (required)
pack_restart	pack atom quantities into a buffer (required)
unpack_restart	unpack atom quantities from a buffer (required)
create_atom	create an individual atom of this style (required)
data_atom	parse an atom line from the data file (required)
data_atom_hybrid	parse additional atom info unique to this atom style (optional)
data_vel	parse one line of velocity information from data file (optional)
data_vel_hybrid	parse additional velocity data unique to this atom style (optional)
memory_usage	tally memory allocated by atom arrays (required)

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

IMPORTANT NOTE: It is possible to add some attributes, such as a molecule ID, to atom styles that do not have them via the fix property/atom command. This command also allows new custom attributes consisting of extra integer or floating-point values to be added to atoms. See the fix property/atom doc page for examples of cases where this is useful and details on how to initialize, access, and output the custom values.

New pair styles, fixes, or computes can be added to LIGGGHTS(R)-WSU, as discussed below. The code for these classes can use the per-atom properties defined by fix property/atom. The Atom class has a find_custom() method that is useful in this context:

```
int index = atom->find_custom(char *name, int &flag);
```

The "name" of a custom attribute, as specified in the fix property/atom command, is checked to verify that it exists and its index is returned. The method also sets flag = 0/1 depending on whether it is an integer or floating-point attribute. The vector of values associated with the attribute can then be accessed using the returned index as

```
int *ivector = atom->ivector[index];
```

8.1 Atom styles   555

```
double *dvector = atom->dvector[index];
```

Ivector or dvector are vectors of length Nlocal = # of owned atoms, which store the attributes of individual atoms.

8.2 Compute styles

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

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

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

init	perform one time setup (required)
init_list	neighbor list setup, if needed (optional)
compute_scalar	compute a scalar quantity (optional)
compute_vector	compute a vector of quantities (optional)
compute_peratom	compute one or more quantities per atom (optional)
compute_local	compute one or more quantities per processor (optional)
pack_comm	pack a buffer with items to communicate (optional)
unpack_comm	unpack the buffer (optional)
pack_reverse	pack a buffer with items to reverse communicate (optional)
unpack_reverse	unpack the buffer (optional)
remove_bias	remove velocity bias from one atom (optional)
remove_bias_all	remove velocity bias from all atoms in group (optional)
restore_bias	restore velocity bias for one atom after remove_bias (optional)
restore_bias_all	same as before, but for all atoms in group (optional)
memory_usage	tally memory usage (optional)

8.3 Dump styles

8.4 Dump custom output options

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

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

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

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

write_data	write a proc's data to a file

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

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

8.5 Fix styles

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

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

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

setmask	determines when the fix is called during the timestep (required)
init	initialization before a run (optional)
setup_pre_exchange	called before atom exchange in setup (optional)
setup_pre_force	called before force computation in setup (optional)
setup	called immediately before the 1st timestep and after forces are computed (optional)
min_setup_pre_force	like setup_pre_force, but for minimizations instead of MD runs (optional)
min_setup	like setup, but for minimizations instead of MD runs (optional)
initial_integrate	called at very beginning of each timestep (optional)
pre_exchange	called before atom exchange on re-neighboring steps (optional)
pre_neighbor	called before neighbor list build (optional)
pre_force	called after pair & molecular forces are computed (optional)
post_force	called after pair & molecular forces are computed and communicated (optional)
final_integrate	called at end of each timestep (optional)
end_of_step	called at very end of timestep (optional)
write_restart	dumps fix info to restart file (optional)
restart	uses info from restart file to re-initialize the fix (optional)
grow_arrays	allocate memory for atom-based arrays used by fix (optional)
copy_arrays	copy atom info when an atom migrates to a new processor (optional)
pack_exchange	store atom's data in a buffer (optional)
unpack_exchange	retrieve atom's data from a buffer (optional)
pack_restart	store atom's data for writing to restart file (optional)
unpack_restart	retrieve atom's data from a restart file buffer (optional)
size_restart	size of atom's data (optional)
maxsize_restart	max size of atom's data (optional)
setup_pre_force_respa	same as setup_pre_force, but for rRESPA (optional)

initial_integrate_respa	same as initial_integrate, but for rRESPA (optional)
post_integrate_respa	called after the first half integration step is done in rRESPA (optional)
pre_force_respa	same as pre_force, but for rRESPA (optional)
post_force_respa	same as post_force, but for rRESPA (optional)
final_integrate_respa	same as final_integrate, but for rRESPA (optional)
min_pre_force	called after pair & molecular forces are computed in minimizer (optional)
min_post_force	called after pair & molecular forces are computed and communicated in minmizer (optional)
min_store	store extra data for linesearch based minimization on a LIFO stack (optional)
min_pushstore	push the minimization LIFO stack one element down (optional)
min_popstore	pop the minimization LIFO stack one element up (optional)
min_clearstore	clear minimization LIFO stack (optional)
min_step	reset or move forward on line search minimization (optional)
min_dof	report number of degrees of freedom *added* by this fix in minimization (optional)
max_alpha	report maximum allowed step size during linesearch minimization (optional)
pack_comm	pack a buffer to communicate a per-atom quantity (optional)
unpack_comm	unpack a buffer to communicate a per-atom quantity (optional)
pack_reverse_comm	pack a buffer to reverse communicate a per-atom quantity (optional)
unpack_reverse_comm	unpack a buffer to reverse communicate a per-atom quantity (optional)
dof	report number of degrees of freedom *removed* by this fix during MD (optional)
compute_scalar	return a global scalar property that the fix computes (optional)
compute_vector	return a component of a vector property that the fix computes (optional)
compute_array	return a component of an array property that the fix computes (optional)
deform	called when the box size is changed (optional)
reset_target	called when a change of the target temperature is requested during a run (optional)
reset_dt	is called when a change of the time step is requested during a run (optional)
modify_param	called when a fix_modify request is executed (optional)
memory_usage	report memory used by fix (optional)
thermo	compute quantities for thermodynamic output (optional)

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

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

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

8.6 Input script commands

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

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

command	operations performed by the new command

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

8.7 Pairwise potentials

Classes that compute pairwise interactions are derived from the Pair class. In LIGGGHTS(R)-WSU, pairwise calculation include manybody potentials such as EAM or Tersoff where particles interact without a static bond topology. New styles can be created to add new pair potentials to LIGGGHTS(R)-WSU.

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

compute	workhorse routine that computes pairwise interactions
settings	reads the input script line with arguments you define
coeff	set coefficients for one i,j type pair
init_one	perform initialization for one i,j type pair
init_style	initialization specific to this pair style
write & read_restart	write/read i,j pair coeffs to restart files
write & read_restart_settings	write/read global settings to restart files
single	force and energy of a single pairwise interaction between 2 atoms
compute_inner/middle/outer	versions of compute used by rRESPA

The inner/middle/outer routines are optional.

8.8 Region styles

Classes that define geometric regions are derived from the Region class. Regions are used elsewhere in LIGGGHTS(R)-WSU to group atoms, delete atoms to create a void, insert atoms in a specified region, etc. New styles can be created to add new region shapes to LIGGGHTS(R)-WSU.

Region_sphere.cpp is an example of a spherical region.

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

match	determine whether a point is in the region

8.9 Thermodynamic output options

There is one class that computes and prints thermodynamic information to the screen and log file; see the file thermo.cpp.

There are two styles defined in thermo.cpp: "one" and "multi". There is also a flexible "custom" style which allows the user to explicitly list keywords for quantities to print when thermodynamic info is output. See the thermo_style command for a list of defined quantities.

The thermo styles (one, multi, etc) are simply lists of keywords. Adding a new style thus only requires defining a new list of keywords. Search for the word "customize" with references to "thermo style" in thermo.cpp to see the two locations where code will need to be added.

New keywords can also be added to thermo.cpp to compute new quantities for output. Search for the word "customize" with references to "keyword" in thermo.cpp to see the several locations where code will need to be added.

Note that the thermo_style custom command already allows for thermo output of quantities calculated by fixes, computes, and variables. Thus, it may be simpler to compute what you wish via one of those constructs, than by adding a new keyword to the thermo command.

8.10 Variable options

There is one class that computes and stores variable information in LIGGGHTS(R)-WSU; see the file variable.cpp. The value associated with a variable can be periodically printed to the screen via the print, fix print, or thermo_style custom commands. Variables of style "equal" can compute complex equations that involve the following types of arguments:

thermo keywords = ke, vol, atoms, ... other variables = v_a, v_myvar, ... math functions = div(x,y), mult(x,y), add(x,y), ... group functions = mass(group), xcm(group,x), ... atom values = x**123**, y**3**, vx**34**, ... compute values = c_mytemp**0**, c_thermo_press**3**, ...

Adding keywords for the thermo_style custom command (which can then be accessed by variables) was discussed here on this page.

Adding a new math function of one or two arguments can be done by editing one section of the Variable::evaulate() method. Search for the word "customize" to find the appropriate location.

Adding a new group function can be done by editing one section of the Variable::evaulate() method. Search for the word "customize" to find the appropriate location. You may need to add a new method to the Group class as well (see the group.cpp file).

Accessing a new atom-based vector can be done by editing one section of the Variable::evaulate() method. Search for the word "customize" to find the appropriate location.

Adding new compute styles (whose calculated values can then be accessed by variables) was discussed here on this page.

8.11 Submitting new features for inclusion in LIGGGHTS(R)-WSU

We encourage users to submit new features that they add to LIGGGHTS(R)-WSU, especially if you think the features will be of interest to other users. If they are broadly useful we may add them as core files to LIGGGHTS(R)-WSU.

The previous sections of this doc page describe how to add new features of various kinds to LIGGGHTS(R)-WSU. Packages are simply collections of one or more new class files which are invoked as a

new "style" within a LIGGGHTS(R)-WSU input script. If designed correctly, these additions typically do not require changes to the main core of LIGGGHTS(R)-WSU; they are simply add-on files.

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

- All source files you provide must compile with the most current version of LIGGGHTS(R)-WSU.
- If you want your file(s) to be added to main LIGGGHTS(R)-WSU or one of its standard packages, then it needs to be written in a style compatible with other LIGGGHTS(R)-WSU source files. This is so the developers can understand it and hopefully maintain it. This basically means that the code accesses data structures, performs its operations, and is formatted similar to other LIGGGHTS(R)-WSU source files, including the use of the error class for error and warning messages.
- Your new source files need to have the LIGGGHTS(R)-WSU copyright, GPL notice, and your name at the top, like other LIGGGHTS(R)-WSU source files. They need to create a class that is inside the LIGGGHTS(R)-WSU namespace. I.e. they do not need to be in the same stylistic format and syntax as other LIGGGHTS(R)-WSU files, though that would be nice.
- Finally, you must also send a documentation file for each new command or style you are adding to LIGGGHTS(R)-WSU. This will be one file for a single-file feature. For a package, it might be several files. These are simple text files which we will convert to HTML. They must be in the same format as other *.txt files in the lammps/doc directory for similar commands and styles. The txt2html tool we use to do the conversion can be downloaded from this site, so you can perform the HTML conversion yourself to proofread your doc page.

Note that the more clear and self-explanatory you make your doc and README files, the more likely it is that users will try out your new feature.

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

6. Packages

This section gives a quick overview of the add-on packages that extend LIGGGHTS(R)-WSU functionality.

LIGGGHTS(R)-WSU includes optional packages, which are groups of files that enable a specific set of features. For example, force fields for molecular systems or granular systems are in packages. You can see the list of all packages by typing "make package" from within the src directory of the LIGGGHTS(R)-WSU distribution.

See Section_start 3 of the manual for details on how to include/exclude specific packages as part of the LIGGGHTS(R)-WSU build process, and for more details about the differences between standard packages and user packages in LIGGGHTS(R)-WSU.

Below, the packages currently availabe in LIGGGHTS(R)-WSU are listed. For standard packages, just a one-line description is given:

Package	Description	Author(s)	Doc page	Example	Library
ASPHERE	aspherical particles	-	Section_howto	ellipse	-
MOLECULE	molecular system force fields	-	-	-	-
POEMS	coupled rigid body motion	Rudra Mukherjee (JPL)	fix poems	rigid	lib/poems
RIGID	rigid bodies	-	fix rigid	rigid	-
VORONOI	Voronoi tesselations	Daniel Schwen (LANL)	compute voronoi/atom	-	Voro++

The "Doc page" column links to either a portion of the Section_howto of the manual, or an input script command implemented as part of the package.

The "Example" column is a sub-directory in the examples directory of the distribution which has an input script that uses the package. E.g. "peptide" refers to the examples/peptide directory.

The "Library" column lists an external library which must be built first and which LIGGGHTS(R)-WSU links to when it is built. If it is listed as lib/package, then the code for the library is under the lib directory of the LIGGGHTS(R)-WSU distribution. See the lib/package/README file for info on how to build the library. If it is not listed as lib/package, then it is a third-party library not included in the LIGGGHTS(R)-WSU distribution. See the src/package/README or src/package/Makefile.lammps file for info on where to download the library. Section start of the manual also gives details on how to build LIGGGHTS(R)-WSU with both kinds of auxiliary libraries.

9. Python interface to LIGGGHTS(R)-WSU

This section describes how to build and use LIGGGHTS(R)-WSU via a Python interface.

- 9.1 Building LIGGGHTS(R)-WSU as a shared library
- 9.2 Installing the Python wrapper into Python
- 9.3 Extending Python with MPI to run in parallel
- 9.4 Testing the Python-LIGGGHTS(R)-WSU interface
- 9.5 Using LIGGGHTS(R)-WSU from Python
- 9.6 Example Python scripts that use LIGGGHTS(R)-WSU

The LIGGGHTS(R)-WSU distribution includes the file python/lammps.py which wraps the library interface to LIGGGHTS(R)-WSU. This file makes it is possible to run LIGGGHTS(R)-WSU, invoke LIGGGHTS(R)-WSU commands or give it an input script, extract LIGGGHTS(R)-WSU results, an modify internal LIGGGHTS(R)-WSU variables, either from a Python script or interactively from a Python prompt. You can do the former in serial or parallel. Running Python interactively in parallel does not generally work, unless you have a package installed that extends your Python to enable multiple instances of Python to read what you type.

Python is a powerful scripting and programming language which can be used to wrap software like LIGGGHTS(R)-WSU and other packages. It can be used to glue multiple pieces of software together, e.g. to run a coupled or multiscale model. See Section section of the manual and the couple directory of the distribution for more ideas about coupling LIGGGHTS(R)-WSU to other codes. See Section_start 4 about how to build LIGGGHTS(R)-WSU as a library, and Section_howto 19 for a description of the library interface provided in src/library.cpp and src/library.h and how to extend it for your needs. As described below, that interface is what is exposed to Python. It is designed to be easy to add functions to. This can easily extend the Python inteface as well. See details below.

By using the Python interface, LIGGGHTS(R)-WSU can also be coupled with a GUI or other visualization tools that display graphs or animations in real time as LIGGGHTS(R)-WSU runs. Examples of such scripts are inlcluded in the python directory.

Two advantages of using Python are how concise the language is, and that it can be run interactively, enabling rapid development and debugging of programs. If you use it to mostly invoke costly operations within LIGGGHTS(R)-WSU, such as running a simulation for a reasonable number of timesteps, then the overhead cost of invoking LIGGGHTS(R)-WSU thru Python will be negligible.

Before using LIGGGHTS(R)-WSU from a Python script, you need to do two things. You need to build LIGGGHTS(R)-WSU as a dynamic shared library, so it can be loaded by Python. And you need to tell Python how to find the library and the Python wrapper file python/lammps.py. Both these steps are discussed below. If you wish to run LIGGGHTS(R)-WSU in parallel from Python, you also need to extend your Python with MPI. This is also discussed below.

The Python wrapper for LIGGGHTS(R)-WSU uses the amazing and magical (to me) "ctypes" package in Python, which auto-generates the interface code needed between Python and a set of C interface routines for a library. Ctypes is part of standard Python for versions 2.5 and later. You can check which version of Python you have installed, by simply typing "python" at a shell prompt.

9.1 Building LIGGGHTS(R)-WSU as a shared library

Instructions on how to build LIGGGHTS(R)-WSU as a shared library are given in Section_start 5. A shared library is one that is dynamically loadable, which is what Python requires. On Linux this is a library file that ends in ".so", not ".a".

From the src directory, type

```
make makeshlib
make −f Makefile.shlib foo
```

where foo is the machine target name, such as linux or g++ or serial. This should create the file liblammps_foo.so in the src directory, as well as a soft link liblammps.so, which is what the Python wrapper will load by default. Note that if you are building multiple machine versions of the shared library, the soft link is always set to the most recently built version.

If this fails, see Section_start 5 for more details, especially if your LIGGGHTS(R)-WSU build uses auxiliary libraries like MPI or FFTW which may not be built as shared libraries on your system.

9.2 Installing the Python wrapper into Python

For Python to invoke LIGGGHTS(R)-WSU, there are 2 files it needs to know about:

- python/lammps.py
- src/liblammps.so

Lammps.py is the Python wrapper on the LIGGGHTS(R)-WSU library interface. Liblammps.so is the shared LIGGGHTS(R)-WSU library that Python loads, as described above.

You can insure Python can find these files in one of two ways:

- set two environment variables
- run the python/install.py script

If you set the paths to these files as environment variables, you only have to do it once. For the csh or tcsh shells, add something like this to your ~/.cshrc file, one line for each of the two files:

```
setenv PYTHONPATH $PYTHONPATH:/home/sjplimp/lammps/python
setenv LD_LIBRARY_PATH $LD_LIBRARY_PATH:/home/sjplimp/lammps/src
```

If you use the python/install.py script, you need to invoke it every time you rebuild LIGGGHTS(R)-WSU (as a shared library) or make changes to the python/lammps.py file.

You can invoke install.py from the python directory as

```
% python install.py [libdir] [pydir]
```

The optional libdir is where to copy the LIGGGHTS(R)-WSU shared library to; the default is /usr/local/lib. The optional pydir is where to copy the lammps.py file to; the default is the site-packages directory of the version of Python that is running the install script.

Note that libdir must be a location that is in your default LD_LIBRARY_PATH, like /usr/local/lib or /usr/lib. And pydir must be a location that Python looks in by default for imported modules, like its site-packages dir. If you want to copy these files to non-standard locations, such as within your own user space, you will need to

set your PYTHONPATH and LD_LIBRARY_PATH environment variables accordingly, as above.

If the install.py script does not allow you to copy files into system directories, prefix the python command with "sudo". If you do this, make sure that the Python that root runs is the same as the Python you run. E.g. you may need to do something like

```
% sudo /usr/local/bin/python install.py [libdir] [pydir]
```

You can also invoke install.py from the make command in the src directory as

```
% make install-python
```

In this mode you cannot append optional arguments. Again, you may need to prefix this with "sudo". In this mode you cannot control which Python is invoked by root.

Note that if you want Python to be able to load different versions of the LIGGGHTS(R)-WSU shared library (see this section below), you will need to manually copy files like liblammps_g++.so into the appropriate system directory. This is not needed if you set the LD_LIBRARY_PATH environment variable as described above.

9.3 Extending Python with MPI to run in parallel

If you wish to run LIGGGHTS(R)-WSU in parallel from Python, you need to extend your Python with an interface to MPI. This also allows you to make MPI calls directly from Python in your script, if you desire.

There are several Python packages available that purport to wrap MPI as a library and allow MPI functions to be called from Python.

These include

- pyMPI
- maroonmpi
- mpi4py
- myMPI
- Pypar

All of these except pyMPI work by wrapping the MPI library and exposing (some portion of) its interface to your Python script. This means Python cannot be used interactively in parallel, since they do not address the issue of interactive input to multiple instances of Python running on different processors. The one exception is pyMPI, which alters the Python interpreter to address this issue, and (I believe) creates a new alternate executable (in place of "python" itself) as a result.

In principle any of these Python/MPI packages should work to invoke LIGGGHTS(R)-WSU in parallel and MPI calls themselves from a Python script which is itself running in parallel. However, when I downloaded and looked at a few of them, their documentation was incomplete and I had trouble with their installation. It's not clear if some of the packages are still being actively developed and supported.

The one I recommend, since I have successfully used it with LIGGGHTS(R)-WSU, is Pypar. Pypar requires the ubiquitous Numpy package be installed in your Python. After launching python, type

```
import numpy
```

to see if it is installed. If not, here is how to install it (version 1.3.0b1 as of April 2009). Unpack the numpy tarball and from its top-level directory, type

```
python setup.py build
sudo python setup.py install
```

The "sudo" is only needed if required to copy Numpy files into your Python distribution's site-packages directory.

To install Pypar (version pypar-2.1.4_94 as of Aug 2012), unpack it and from its "source" directory, type

```
python setup.py build
sudo python setup.py install
```

Again, the "sudo" is only needed if required to copy Pypar files into your Python distribution's site-packages directory.

If you have successully installed Pypar, you should be able to run Python and type

```
import pypar
```

without error. You should also be able to run python in parallel on a simple test script

```
% mpirun -np 4 python test.py
```

where test.py contains the lines

```
import pypar
print "Proc %d out of %d procs" % (pypar.rank(),pypar.size())
```

and see one line of output for each processor you run on.

IMPORTANT NOTE: To use Pypar and LIGGGHTS(R)-WSU in parallel from Python, you must insure both are using the same version of MPI. If you only have one MPI installed on your system, this is not an issue, but it can be if you have multiple MPIs. Your LIGGGHTS(R)-WSU build is explicit about which MPI it is using, since you specify the details in your lo-level src/MAKE/Makefile.foo file. Pypar uses the "mpicc" command to find information about the MPI it uses to build against. And it tries to load "libmpi.so" from the LD_LIBRARY_PATH. This may or may not find the MPI library that LIGGGHTS(R)-WSU is using. If you have problems running both Pypar and LIGGGHTS(R)-WSU together, this is an issue you may need to address, e.g. by moving other MPI installations so that Pypar finds the right one.

9.4 Testing the Python-LIGGGHTS(R)-WSU interface

To test if LIGGGHTS(R)-WSU is callable from Python, launch Python interactively and type:

```
>>> from lammps import lammps
>>> lmp = lammps()
```

If you get no errors, you're ready to use LIGGGHTS(R)-WSU from Python. If the 2nd command fails, the most common error to see is

```
OSError: Could not load LIGGGHTS(R)-WSU dynamic library
```

which means Python was unable to load the LIGGGHTS(R)-WSU shared library. This typically occurs if the system can't find the LIGGGHTS(R)-WSU shared library or one of the auxiliary shared libraries it depends on, or if something about the library is incompatible with your Python. The error message should give you an indication of what went wrong.

You can also test the load directly in Python as follows, without first importing from the lammps.py file:

```
>>> from ctypes import CDLL
>>> CDLL("liblammps.so")
```

If an error occurs, carefully go thru the steps in <u>Section start 5</u> and above about building a shared library and about insuring Python can find the necessary two files it needs.

Test LIGGGHTS(R)-WSU and Python in serial:

To run a LIGGGHTS(R)-WSU test in serial, type these lines into Python interactively from the bench directory:

```
>>> from lammps import lammps
>>> lmp = lammps()
>>> lmp.file("in.lj")
```

Or put the same lines in the file test.py and run it as

```
% python test.py
```

Either way, you should see the results of running the in.lj benchmark on a single processor appear on the screen, the same as if you had typed something like:

```
lmp_g++ <in.lj
```

Test LIGGGHTS(R)-WSU and Python in parallel:

To run LIGGGHTS(R)-WSU in parallel, assuming you have installed the <u>Pypar</u> package as discussed above, create a test.py file containing these lines:

```
import pypar
from lammps import lammps
lmp = lammps()
lmp.file("in.lj")
print "Proc %d out of %d procs has" % (pypar.rank(),pypar.size()),lmp
pypar.finalize()
```

You can then run it in parallel as:

```
% mpirun -np 4 python test.py
```

and you should see the same output as if you had typed

```
% mpirun -np 4 lmp_g++ <in.lj
```

Note that if you leave out the 3 lines from test.py that specify Pypar commands you will instantiate and run LIGGGHTS(R)-WSU independently on each of the P processors specified in the mpirun command. In this case you should get 4 sets of output, each showing that a LIGGGHTS(R)-WSU run was made on a single processor, instead of one set of output showing that LIGGGHTS(R)-WSU ran on 4 processors. If the 1-processor outputs occur, it means that Pypar is not working correctly.

Also note that once you import the PyPar module, Pypar initializes MPI for you, and you can use MPI calls directly in your Python script, as described in the Pypar documentation. The last line of your Python script should be pypar.finalize(), to insure MPI is shut down correctly.

Running Python scripts:

Note that any Python script (not just for LIGGGHTS(R)-WSU) can be invoked in one of several ways:

```
% python foo.script
% python -i foo.script
% foo.script
```

The last command requires that the first line of the script be something like this:

```
#!/usr/local/bin/python
#!/usr/local/bin/python -i
```

where the path points to where you have Python installed, and that you have made the script file executable:

```
% chmod +x foo.script
```

Without the "-i" flag, Python will exit when the script finishes. With the "-i" flag, you will be left in the Python interpreter when the script finishes, so you can type subsequent commands. As mentioned above, you can only run Python interactively when running Python on a single processor, not in parallel.

9.5 Using LIGGGHTS(R)-WSU from Python

The Python interface to LIGGGHTS(R)-WSU consists of a Python "lammps" module, the source code for which is in python/lammps.py, which creates a "lammps" object, with a set of methods that can be invoked on that object. The sample Python code below assumes you have first imported the "lammps" module in your Python script, as follows:

```
from lammps import lammps
```

These are the methods defined by the lammps module. If you look at the file src/library.cpp you will see that they correspond one-to-one with calls you can make to the LIGGGHTS(R)-WSU library from a C++ or C or Fortran program.

```
lmp = lammps()              # create a LIGGGHTS(R)-WSU object using the default liblammps.so library
lmp = lammps("g++")         # create a LIGGGHTS(R)-WSU object using the liblammps_g++.so library
lmp = lammps("",list)       # ditto, with command-line args, e.g. list = ["-echo","screen"]
lmp = lammps("g++",list)

lmp.close()                 # destroy a LIGGGHTS(R)-WSU object

lmp.file(file)              # run an entire input script, file = "in.lj"
lmp.command(cmd)            # invoke a single LIGGGHTS(R)-WSU command, cmd = "run 100"

xlo = lmp.extract_global(name,type)  # extract a global quantity
                                     # name = "boxxlo", "nlocal", etc
                                     # type = 0 = int
                                     #        1 = double

coords = lmp.extract_atom(name,type)     # extract a per-atom quantity
                                         # name = "x", "type", etc
                                         # type = 0 = vector of ints
                                         #        1 = array of ints
                                         #        2 = vector of doubles
                                         #        3 = array of doubles

eng = lmp.extract_compute(id,style,type)  # extract value(s) from a compute
```

```
v3 = lmp.extract_fix(id,style,type,i,j)      # extract value(s) from a fix
                                             # id = ID of compute or fix
                                             # style = 0 = global data
                                             #         1 = per-atom data
                                             #         2 = local data
                                             # type = 0 = scalar
                                             #        1 = vector
                                             #        2 = array
                                             # i,j = indices of value in global vector or array

var = lmp.extract_variable(name,group,flag)  # extract value(s) from a variable
                                             # name = name of variable
                                             # group = group ID (ignored for equal-style variable
                                             # flag = 0 = equal-style variable
                                             #        1 = atom-style variable

natoms = lmp.get_natoms()                    # total # of atoms as int
data = lmp.gather_atoms(name,type,count)     # return atom attribute of all atoms gathered into data
                                             # name = "x", "charge", "type", etc
                                             # count = # of per-atom values, 1 or 3, etc
lmp.scatter_atoms(name,type,count,data)      # scatter atom attribute of all atoms from data, ordere
                                             # name = "x", "charge", "type", etc
                                             # count = # of per-atom values, 1 or 3, etc
```

IMPORTANT NOTE: Currently, the creation of a LIGGGHTS(R)-WSU object from within lammps.py does not take an MPI communicator as an argument. There should be a way to do this, so that the LIGGGHTS(R)-WSU instance runs on a subset of processors if desired, but I don't know how to do it from Pypar. So for now, it runs with MPI_COMM_WORLD, which is all the processors. If someone figures out how to do this with one or more of the Python wrappers for MPI, like Pypar, please let us know and we will amend these doc pages.

Note that you can create multiple LIGGGHTS(R)-WSU objects in your Python script, and coordinate and run multiple simulations, e.g.

```
from lammps import lammps
lmp1 = lammps()
lmp2 = lammps()
lmp1.file("in.file1")
lmp2.file("in.file2")
```

The file() and command() methods allow an input script or single commands to be invoked.

The extract_global(), extract_atom(), extract_compute(), extract_fix(), and extract_variable() methods return values or pointers to data structures internal to LIGGGHTS(R)-WSU.

For extract_global() see the src/library.cpp file for the list of valid names. New names could easily be added. A double or integer is returned. You need to specify the appropriate data type via the type argument.

For extract_atom(), a pointer to internal LIGGGHTS(R)-WSU atom-based data is returned, which you can use via normal Python subscripting. See the extract() method in the src/atom.cpp file for a list of valid names. Again, new names could easily be added. A pointer to a vector of doubles or integers, or a pointer to an array of doubles (double **) or integers (int **) is returned. You need to specify the appropriate data type via the type argument.

For extract_compute() and extract_fix(), the global, per-atom, or local data calulated by the compute or fix can be accessed. What is returned depends on whether the compute or fix calculates a scalar or vector or array. For a scalar, a single double value is returned. If the compute or fix calculates a vector or array, a pointer to the internal LIGGGHTS(R)-WSU data is returned, which you can use via normal Python subscripting. The one exception is that for a fix that calculates a global vector or array, a single double value

from the vector or array is returned, indexed by I (vector) or I and J (array). I,J are zero-based indices. The I,J arguments can be left out if not needed. See Section_howto 15 of the manual for a discussion of global, per-atom, and local data, and of scalar, vector, and array data types. See the doc pages for individual computes and fixes for a description of what they calculate and store.

For extract_variable(), an equal-style or atom-style variable is evaluated and its result returned.

For equal-style variables a single double value is returned and the group argument is ignored. For atom-style variables, a vector of doubles is returned, one value per atom, which you can use via normal Python subscripting. The values will be zero for atoms not in the specified group.

The get_natoms() method returns the total number of atoms in the simulation, as an int.

The gather_atoms() method returns a ctypes vector of ints or doubles as specified by type, of length count*natoms, for the property of all the atoms in the simulation specified by name, ordered by count and then by atom ID. The vector can be used via normal Python subscripting. If atom IDs are not consecutively ordered within LIGGGHTS(R)-WSU, a None is returned as indication of an error.

Note that the data structure gather_atoms("x") returns is different from the data structure returned by extract_atom("x") in four ways. (1) Gather_atoms() returns a vector which you index as x[i]; extract_atom() returns an array which you index as x[i][j]. (2) Gather_atoms() orders the atoms by atom ID while extract_atom() does not. (3) Gathert_atoms() returns a list of all atoms in the simulation; extract_atoms() returns just the atoms local to each processor. (4) Finally, the gather_atoms() data structure is a copy of the atom coords stored internally in LIGGGHTS(R)-WSU, whereas extract_atom() returns an array that effectively points directly to the internal data. This means you can change values inside LIGGGHTS(R)-WSU from Python by assigning a new values to the extract_atom() array. To do this with the gather_atoms() vector, you need to change values in the vector, then invoke the scatter_atoms() method.

The scatter_atoms() method takes a vector of ints or doubles as specified by type, of length count*natoms, for the property of all the atoms in the simulation specified by name, ordered by bount and then by atom ID. It uses the vector of data to overwrite the corresponding properties for each atom inside LIGGGHTS(R)-WSU. This requires LIGGGHTS(R)-WSU to have its "map" option enabled; see the atom_modify command for details. If it is not, or if atom IDs are not consecutively ordered, no coordinates are reset.

The array of coordinates passed to scatter_atoms() must be a ctypes vector of ints or doubles, allocated and initialized something like this:

```
from ctypes import *
natoms = lmp.get_natoms()
n3 = 3*natoms
x = (n3*c_double)()
x0 = x coord of atom with ID 1
x1 = y coord of atom with ID 1
x2 = z coord of atom with ID 1
x3 = x coord of atom with ID 2
...
xn3-1 = z coord of atom with ID natoms
lmp.scatter_coords("x",1,3,x)
```

Alternatively, you can just change values in the vector returned by gather_atoms("x",1,3), since it is a ctypes vector of doubles.

As noted above, these Python class methods correspond one-to-one with the functions in the LIGGGHTS(R)-WSU library interface in src/library.cpp and library.h. This means you can extend the Python wrapper via the following steps:

- Add a new interface function to src/library.cpp and src/library.h.
- Rebuild LIGGGHTS(R)-WSU as a shared library.
- Add a wrapper method to python/lammps.py for this interface function.
- You should now be able to invoke the new interface function from a Python script. Isn't ctypes amazing?

9.6 Example Python scripts that use LIGGGHTS(R)-WSU

These are the Python scripts included as demos in the python/examples directory of the LIGGGHTS(R)-WSU distribution, to illustrate the kinds of things that are possible when Python wraps LIGGGHTS(R)-WSU. If you create your own scripts, send them to us and we can include them in the LIGGGHTS(R)-WSU distribution.

trivial.py	read/run a LIGGGHTS(R)-WSU input script thru Python
demo.py	invoke various LIGGGHTS(R)-WSU library interface routines
simple.py	mimic operation of couple/simple/simple.cpp in Python
gui.py	GUI go/stop/temperature-slider to control LIGGGHTS(R)-WSU
plot.py	real-time temeperature plot with GnuPlot via Pizza.py
viz_tool.py	real-time viz via some viz package
vizplotgui_tool.py	combination of viz_tool.py and plot.py and gui.py

For the viz_tool.py and vizplotgui_tool.py commands, replace "tool" with "gl" or "atomeye" or "pymol" or "vmd", depending on what visualization package you have installed.

Note that for GL, you need to be able to run the Pizza.py GL tool, which is included in the pizza sub-directory. See the Pizza.py doc pages for more info:

Note that for AtomEye, you need version 3, and there is a line in the scripts that specifies the path and name of the executable. See the AtomEye WWW pages here or here for more details:

```
http://mt.seas.upenn.edu/Archive/Graphics/A
http://mt.seas.upenn.edu/Archive/Graphics/A3/A3.html
```

The latter link is to AtomEye 3 which has the scriping capability needed by these Python scripts.

Note that for PyMol, you need to have built and installed the open-source version of PyMol in your Python, so that you can import it from a Python script. See the PyMol WWW pages here or here for more details:

```
http://www.pymol.org
http://sourceforge.net/scm/?type=svn&group_id=4546
```

The latter link is to the open-source version.

See the python/README file for instructions on how to run them and the source code for individual scripts for comments about what they do.

Here are screenshots of the vizplotgui_tool.py script in action for different visualization package options. Click to see larger images:
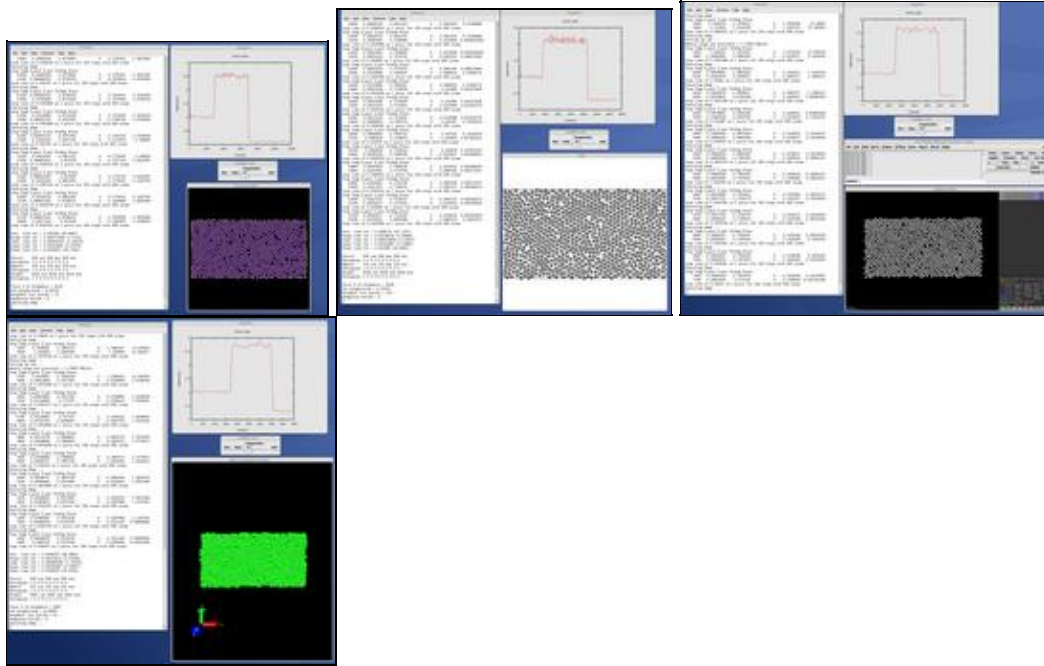
2. Getting Started

This section describes how to build and run LIGGGHTS(R)-WSU, for both new and experienced users.

2.1 What's in the LIGGGHTS(R)-WSU distribution
2.2 Making LIGGGHTS(R)-WSU
2.3 Making LIGGGHTS(R)-WSU with optional packages
2.4 Building LIGGGHTS(R)-WSU via the Make.py script
2.5 Building LIGGGHTS(R)-WSU as a library
2.6 Running LIGGGHTS(R)-WSU
2.7 Command-line options
2.8 Screen output

2.1 What's in the LIGGGHTS(R)-WSU distribution

When you download LIGGGHTS(R)-WSU 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 LIGGGHTS(R)-WSU directory containing two files and several sub-directories:

README	text file
LICENSE	the GNU General Public License (GPL)
doc	documentation
examples	simple test problems
src	source files

2.2 Making LIGGGHTS(R)-WSU

This section has the following sub-sections:

- Read this first
- Steps to build a LIGGGHTS(R)-WSU executable
- Common errors that can occur when making LIGGGHTS(R)-WSU
- Additional build tips
- Building for a Mac
- Building for Windows

Read this first:

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

Please read this section carefully. If you are not comfortable with makefiles, or building codes on a Unix platform, or running an MPI job on your machine, please find a local expert to help you. Many compiling, linking, and run problems that users have are often not LIGGGHTS(R)-WSU issues - they are peculiar to the

user's system, compilers, libraries, etc. Such questions are better answered by a local expert.

If you have a build problem that you are convinced is a LIGGGHTS(R)-WSU issue (e.g. the compiler complains about a line of LIGGGHTS(R)-WSU source code), then please post a question to the LIGGGHTS(R)-WSU mail list.

If you succeed in building LIGGGHTS(R)-WSU on a new kind of machine, for which there isn't a similar Makefile for in the src/MAKE directory, send it to the developers and we can include it in the LIGGGHTS(R)-WSU distribution.

Steps to build a LIGGGHTS(R)-WSU executable:

Step 0

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

```
make linux
or
gmake mac
```

Note that on a multi-processor or multi-core platform you can launch a parallel make, by using the "-j" switch with the make command, which will build LIGGGHTS(R)-WSU more quickly.

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

Note that by default only a few of LIGGGHTS(R)-WSU optional packages are installed. To build LIGGGHTS(R)-WSU with optional packages, see this section below.

Step 1

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

Step 2

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

Step 3

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

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

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

Step 4

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

The LMP_INC variable is used to include options that turn on ifdefs within the LIGGGHTS(R)-WSU code. The options that are currently recogized are:

- -DLAMMPS_GZIP
- -DLAMMPS_JPEG
- -DLAMMPS_PNG
- -DLAMMPS_FFMPEG
- -DLAMMPS_MEMALIGN
- -DLAMMPS_XDR
- -DLAMMPS_SMALLBIG
- -DLAMMPS_BIGBIG
- -DLAMMPS_SMALLSMALL
- -DLAMMPS_LONGLONG_TO_LONG
- -DPACK_ARRAY
- -DPACK_POINTER
- -DPACK_MEMCPY

The read_data and dump commands will read/write gzipped files if you compile with -DLAMMPS_GZIP. It requires that your machine supports the "popen" function in the standard runtime library and that a gzip executable can be found by LIGGGHTS(R)-WSU during a run.

If you use -DLAMMPS_JPEG, the dump image command will be able to write out JPEG image files. For JPEG files, you must also link LIGGGHTS(R)-WSU with a JPEG library, as described below. If you use -DLAMMPS_PNG, the dump image command will be able to write out PNG image files. For PNG files, you must also link LIGGGHTS(R)-WSU with a PNG library, as described below. If neither of those two defines are used, LIGGGHTS(R)-WSU will only be able to write out uncompressed PPM image files.

If you use -DLAMMPS_FFMPEG, the dump movie command will be available to support on-the-fly generation of rendered movies the need to store intermediate image files. It requires that your machines supports the "popen" function in the standard runtime library and that an FFmpeg executable can be found by LIGGGHTS(R)-WSU during the run.

Using -DLAMMPS_MEMALIGN= enables the use of the posix_memalign() call instead of malloc() when large chunks or memory are allocated by LIGGGHTS(R)-WSU. This can help to make more efficient use of vector instructions of modern CPUS, since dynamically allocated memory has to be aligned on larger than default byte boundaries (e.g. 16 bytes instead of 8 bytes on x86 type platforms) for optimal performance.

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

Use at most one of the -DLAMMPS_SMALLBIG, -DLAMMPS_BIGBIG, -D-DLAMMPS_SMALLSMALL settings. The default is -DLAMMPS_SMALLBIG. These settings refer to use of 4-byte (small) vs 8-byte (big) integers within LIGGGHTS(R)-WSU, as specified in src/lmptype.h. The only reason to use the BIGBIG setting is to enable simulation of huge molecular systems with more than 2 billion atoms or to allow moving atoms to wrap back through a periodic box more than 512 times. The only reason to use the SMALLSMALL setting is if your machine does not support 64-bit integers. See the Additional build tips section below for more details.

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

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

Step 5

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

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

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

If you are installing MPI yourself, we recommend Argonne's MPICH2 or OpenMPI. MPICH can be downloaded from the Argonne MPI site. OpenMPI can be downloaded from the OpenMPI site. Other MPI packages 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 is likely to be faster than a self-installed MPICH or OpenMPI, so find out how to build and link with it. If you use MPICH or OpenMPI, 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 LIGGGHTS(R)-WSU build, which can avoid problems that can arise when linking LIGGGHTS(R)-WSU to the MPI library.

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

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

Step 6

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

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

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

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

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

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

Step 7

The 3 JPG variables allow you to specify a JPEG and/or PNG library which LIGGGHTS(R)-WSU uses when writing out JPEG or PNG files via the dump image command. These can be left blank if you do not use the -DLAMMPS_JPEG or -DLAMMPS_PNG switches discussed above in Step 4, since in that case JPEG/PNG output will be disabled.

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

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

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

these variables.

Step 8

Note that by default only a few of LIGGGHTS(R)-WSU optional packages are installed. To build LIGGGHTS(R)-WSU with optional packages, see this section below, before proceeding to Step 9.

Step 9

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

```
make foo
or
gmake foo
```

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

Errors that can occur when making LIGGGHTS(R)-WSU:

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

Here are two non-obvious errors that can occur:

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

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

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

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

Additional build tips:

(1) Building LIGGGHTS(R)-WSU for multiple platforms.

You can make LIGGGHTS(R)-WSU for multiple platforms from the same src directory. Each target creates its own object sub-directory called Obj_target where it stores the system-specific *.o files.

(2) Cleaning up.

Typing "make clean-all" or "make clean-machine" will delete *.o object files created when LIGGGHTS(R)-WSU is built, for either all builds or for a particular machine.

(3) Changing the LIGGGHTS(R)-WSU size limits via -DLAMMPS_SMALLBIG or -DLAMMPS_BIBIG or -DLAMMPS_SMALLSMALL

As explained above, any of these 3 settings can be specified on the LMP_INC line in your low-level src/MAKE/Makefile.foo.

The default is -DLAMMPS_SMALLBIG which allows for systems with up to 2^63 atoms and timesteps (about 9 billion billion). The atom limit is for atomic systems that do not require atom IDs. For molecular models, which require atom IDs, the limit is 2^31 atoms (about 2 billion). With this setting, image flags are stored in 32-bit integers, which means for 3 dimensions that atoms can only wrap around a periodic box at most 512 times. If atoms move through the periodic box more than this limit, the image flags will "roll over", e.g. from 511 to -512, which can cause diagnostics like the mean-squared displacement, as calculated by the compute msd command, to be faulty.

To allow for larger molecular systems or larger image flags, compile with -DLAMMPS_BIGBIG. This enables molecular systems with up to 2^63 atoms (about 9 billion billion). And image flags will not "roll over" until they reach 2^20 = 1048576.

IMPORTANT NOTE: As of 6/2012, the BIGBIG setting does not yet enable molecular systems to grow as large as 2^63. Only the image flag roll over is currently affected by this compile option.

If your system does not support 8-byte integers, you will need to compile with the -DLAMMPS_SMALLSMALL setting. This will restrict your total number of atoms (for atomic or molecular models) and timesteps to 2^31 (about 2 billion). Image flags will roll over at 2^9 = 512.

Note that in src/lmptype.h there are also settings for the MPI data types associated with the integers that store atom IDs and total system sizes. These need to be consistent with the associated C data types, or else LIGGGHTS(R)-WSU will generate a run-time error.

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

Building for a Mac:

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

Building for Windows:

The LIGGGHTS(R)-WSU download page has an option to download both a serial and parallel pre-built Windows exeutable. See the Running LIGGGHTS(R)-WSU section for instructions for running these executables on a Windows box.

The pre-built executables are built with a subset of the available pacakges; see the download page for the list. If you want a Windows version with specific packages included and excluded, you can build it yourself.

One way to do this is install and use cygwin to build LIGGGHTS(R)-WSU with a standard Linus make, just as you would on any Linux box; see src/MAKE/Makefile.cygwin.

The other way to do this is using Visual Studio and project files. See the src/WINDOWS directory and its README.txt file for instructions on both a basic build and a customized build with pacakges you select.

2.3 Making LIGGGHTS(R)-WSU with optional packages

This section has the following sub-sections:

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

Package basics:

The source code for LIGGGHTS(R)-WSU is structured as a set of core files which are always included, plus optional packages. Packages are groups of files that enable a specific set of features. For example, force fields for molecular systems or granular systems are in packages. You can see the list of all packages by typing "make package" from within the src directory of the LIGGGHTS(R)-WSU distribution.

If you use a command in a LIGGGHTS(R)-WSU input script that is specific to a particular package, you must have built LIGGGHTS(R)-WSU with that package, else you will get an error that the style is invalid or the command is unknown. Every command's doc page specfies if it is part of a package. You can also type

```
lmp_machine -h
```

to run your executable with the optional -h command-line switch for "help", which will list the styles and commands known to your executable.

There are two kinds of packages in LIGGGHTS(R)-WSU, standard and user packages. More information about the contents of standard and user packages is given in Section_packages of the manual. The difference between standard and user packages is as follows:

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

User packages have been contributed by users, and always begin with the user prefix. If they are a single command (single file), they are typically in the user-misc package. Otherwise, they are a a set of files grouped together which add a specific functionality to the code.

Some packages (both standard and user) require additional libraries. See more details below.

Including/excluding packages:

To use or not use a package you must include or exclude it before building LIGGGHTS(R)-WSU. From the src directory, this is typically as simple as:

```
make yes-asphere
make g++
```

or

```
make no-asphere
make g++
```

IMPORTANT NOTE: You should NOT include/exclude packages and build LIGGGHTS(R)-WSU in a single make command by using multiple targets, e.g. make yes-colloid g++. This is because the make procedure creates a list of source files that will be out-of-date for the build if the package configuration changes during the same command.

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

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

When you download a LIGGGHTS(R)-WSU tarball, these packages are pre-installed in the src directory: KSPACE, MANYBODY,MOLECULE. When you download LIGGGHTS(R)-WSU source files from the SVN or Git repositories, no packages are pre-installed.

Packages are included or excluded by typing "make yes-name" or "make no-name", where "name" is the name of the package in lower-case, e.g. name = asphere for the ASPHERE package. You can also type "make yes-all" or "make no-all" to include/exclude various sets of packages. Type "make package" to see the all of the package-related make options.

IMPORTANT NOTE: Inclusion/exclusion of a package works by simply moving files back and forth between the main src directory and sub-directories with the package name. After you have included or excluded a package, you must re-build LIGGGHTS(R)-WSU.

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

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

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

Packages that require extra libraries:

A few of the standard and user packages require additional auxiliary libraries. They must be compiled first, before LIGGGHTS(R)-WSU is built. If you get a LIGGGHTS(R)-WSU build error about a missing library, this is likely the reason. See the Section_packages doc page for a list of packages that have auxiliary libraries.

Code for some of these auxiliary libraries is included in the LIGGGHTS(R)-WSU distribution under the lib directory. Some auxiliary libraries are not included with LIGGGHTS(R)-WSU; to use the associated package you must download and install the auxiliary library yourself. Example is the VORONOI packages.

For libraries with provided source code, each lib directory has a README file (e.g. lib/reax/README) with instructions on how to build that library. Typically this is done by typing something like:

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

2.3 Making LIGGGHTS(R)-WSU with optional packages   581

If one of the provided Makefiles is not appropriate for your system you will need to edit or add one. Note that all the Makefiles have a setting for EXTRAMAKE at the top that names a Makefile.lammps.* file.

If successful, this will produce 2 files in the lib directory:

```
libpackage.a
Makefile.lammps
```

The Makefile.lammps file is a copy of the EXTRAMAKE file specified in the Makefile you used.

You MUST insure that the settings in Makefile.lammps are appropriate for your system. If they are not, the LIGGGHTS(R)-WSU build will fail.

As explained in the lib/package/README files, they are used to specify additional system libraries and their locations so that LIGGGHTS(R)-WSU can build with the auxiliary library. For example, if the MEAM or REAX packages are used, the auxiliary libraries consist of F90 code, build with a F90 complier. To link that library with LIGGGHTS(R)-WSU (a C++ code) via whatever C++ compiler LIGGGHTS(R)-WSU is built with, typically requires additional Fortran-to-C libraries be included in the link. Another example are the BLAS and LAPACK libraries needed to use the USER-ATC or USER-AWPMD packages.

For libraries without provided source code, see the src/package/Makefile.lammps file for information on where to find the library and how to build it. E.g. the file src/KIM/Makefile.lammps. This file serves the same purpose as the lib/package/Makefile.lammps file described above. It has settings needed when LIGGGHTS(R)-WSU is built to link with the auxiliary library. Again, you MUST insure that the settings in src/package/Makefile.lammps are appropriate for your system and where you installed the auxiliary library. If they are not, the LIGGGHTS(R)-WSU build will fail.

2.4 Building LIGGGHTS(R)-WSU via the Make.py script

The src directory includes a Make.py script, written in Python, which can be used to automate various steps of the build process.

You can run the script from the src directory by typing either:

```
Make.py
python Make.py
```

which will give you info about the tool. For the former to work, you may need to edit the 1st line of the script to point to your local Python. And you may need to insure the script is executable:

```
chmod +x Make.py
```

The following options are supported as switches:

- -i file1 file2 ...
- -p package1 package2 ...
- -u package1 package2 ...
- -e package1 arg1 arg2 package2 ...
- -o dir
- -b machine
- -s suffix1 suffix2 ...
- -l dir
- -j N
- -h switch1 switch2 ...

Help on any switch can be listed by using -h, e.g.

```
Make.py -h -i -p
```

At a hi-level, these are the kinds of package management and build tasks that can be performed easily, using the Make.py tool:

- install/uninstall packages and build the associated external libs (use -p and -u and -e)
- install packages needed for one or more input scripts (use -i and -p)
- build LIGGGHTS(R)-WSU, either in the src dir or new dir (use -b)
- create a new dir with only the source code needed for one or more input scripts (use -i and -o)

The last bullet can be useful when you wish to build a stripped-down version of LIGGGHTS(R)-WSU to run a specific script(s). Or when you wish to move the minimal amount of files to another platform for a remote LIGGGHTS(R)-WSU build.

Note that using Make.py is not a substitute for insuring you have a valid src/MAKE/Makefile.foo for your system, or that external library Makefiles in any lib/* directories you use are also valid for your system. But once you have done that, you can use Make.py to quickly include/exclude the packages and external libraries needed by your input scripts.

2.5 Building LIGGGHTS(R)-WSU as a library

LIGGGHTS(R)-WSU can be built as either a static or shared library, which can then be called from another application or a scripting language. See this section for more info on coupling LIGGGHTS(R)-WSU to other codes. See this section for more info on wrapping and running LIGGGHTS(R)-WSU from Python.

Static library:

To build LIGGGHTS(R)-WSU as a static library (*.a file on Linux), type

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

where foo is the machine name. This kind of library is typically used to statically link a driver application to LIGGGHTS(R)-WSU, so that you can insure all dependencies are satisfied at compile time. Note that inclusion or exclusion of any desired optional packages should be done before typing "make makelib". The first "make" command will create a current Makefile.lib with all the file names in your src dir. The second "make" command will use it to build LIGGGHTS(R)-WSU as a static library, using the ARCHIVE and ARFLAGS settings in src/MAKE/Makefile.foo. The build will create the file liblammps_foo.a which another application can link to.

Shared library:

To build LIGGGHTS(R)-WSU as a shared library (*.so file on Linux), which can be dynamically loaded, e.g. from Python, type

```
make makeshlib
make -f Makefile.shlib foo
```

where foo is the machine name. This kind of library is required when wrapping LIGGGHTS(R)-WSU with Python; see Section_python for details. Again, note that inclusion or exclusion of any desired optional packages should be done before typing "make makelib". The first "make" command will create a current Makefile.shlib with all the file names in your src dir. The second "make" command will use it to build LIGGGHTS(R)-WSU as a shared library, using the SHFLAGS and SHLIBFLAGS settings in src/MAKE/Makefile.foo. The build will create the file liblammps_foo.so which another application can link to

dyamically. It will also create a soft link liblammps.so, which the Python wrapper uses by default.

Note that for a shared library to be usable by a calling program, all the auxiliary libraries it depends on must also exist as shared libraries. This will be the case for libraries included with LIGGGHTS(R)-WSU, such as the dummy MPI library in src/STUBS or any package libraries in lib/packges, since they are always built as shared libraries with the -fPIC switch. However, if a library like MPI or FFTW does not exist as a shared library, the second make command will generate an error. This means you will need to install a shared library version of the package. The build instructions for the library should tell you how to do this.

As an example, here is how to build and install the [MPICH library](#), a popular open-source version of MPI, distributed by Argonne National Labs, as a shared library in the default /usr/local/lib location:

```
./configure --enable-shared
make
make install
```

You may need to use "sudo make install" in place of the last line if you do not have write privileges for /usr/local/lib. The end result should be the file /usr/local/lib/libmpich.so.

Additional requirement for using a shared library:

The operating system finds shared libraries to load at run-time using the environment variable LD_LIBRARY_PATH. So you may wish to copy the file src/liblammps.so or src/liblammps_g++.so (for example) to a place the system can find it by default, such as /usr/local/lib, or you may wish to add the LIGGGHTS(R)-WSU src directory to LD_LIBRARY_PATH, so that the current version of the shared library is always available to programs that use it.

For the csh or tcsh shells, you would add something like this to your ~/.cshrc file:

```
setenv LD_LIBRARY_PATH $LD_LIBRARY_PATH:/home/sjplimp/lammps/src
```

Calling the LIGGGHTS(R)-WSU library:

Either flavor of library (static or shared0 allows one or more LIGGGHTS(R)-WSU objects to be instantiated from the calling program.

When used from a C++ program, all of LIGGGHTS(R)-WSU is wrapped in a LAMMPS_NS namespace; you can safely use any of its classes and methods from within the calling code, as needed.

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

See the sample codes in examples/COUPLE/simple for examples of C++ and C and Fortran codes that invoke LIGGGHTS(R)-WSU thru its library interface. There are other examples as well in the COUPLE directory which are discussed in [Section_howto 10](#) of the manual. See [Section_python](#) of the manual for a description of the Python wrapper provided with LIGGGHTS(R)-WSU that operates through the LIGGGHTS(R)-WSU library interface.

The files src/library.cpp and library.h define the C-style API for using LIGGGHTS(R)-WSU as a library. See [Section_howto 19](#) of the manual for a description of the interface and how to extend it for your needs.

2.6 Running LIGGGHTS(R)-WSU

By default, LIGGGHTS(R)-WSU 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 LIGGGHTS(R)-WSU on any of the sample inputs provided in the examples or bench directory. Input scripts are named in.* and sample outputs are named log.*.name.P where name is a machine and P is the number of processors it was run on.

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

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

On a Windows box, you can skip making LIGGGHTS(R)-WSU and simply download an executable, as described above, though the pre-packaged executables include only certain packages.

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

For the non-MPI version, follow these steps:

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

For the MPI version, which allows you to run LIGGGHTS(R)-WSU under Windows on multiple processors, follow these steps:

- Download and install MPICH2 for Windows.
- You'll need to use the mpiexec.exe and smpd.exe files from the MPICH2 package. Put them in same directory (or path) as the LIGGGHTS(R)-WSU Windows executable.
- Get a command prompt by going to Start->Run... , then typing "cmd".
- Move to the directory where you have saved lmp_win_mpi.exe (e.g. by typing: cd "Documents").
- Then type something like this: "mpiexec -np 4 -localonly lmp_win_mpi -in in.lj", replacing in.lj with the name of your LIGGGHTS(R)-WSU input script.
- Note that you may need to provide smpd with a passphrase --- it doesn't matter what you type.
- In this mode, output may not immediately show up on the screen, so if your input script takes a long time to execute, you may need to be patient before the output shows up.
- Alternatively, you can still use this executable to run on a single processor by typing something like: "lmp_win_mpi -in in.lj".

The screen output from LIGGGHTS(R)-WSU is described in the next section. As it runs, LIGGGHTS(R)-WSU also writes a log.lammps file with the same information.

Note that this sequence of commands copies the LIGGGHTS(R)-WSU executable (lmp_linux) to the directory with the input files. This may not be necessary, but some versions of MPI reset the working directory to where the executable is, rather than leave it as the directory where you launch mpirun from (if you

launch lmp_linux on its own and not under mpirun). If that happens, LIGGGHTS(R)-WSU will look for additional input files and write its output files to the executable directory, rather than your working directory, which is probably not what you want.

If LIGGGHTS(R)-WSU 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 Section_errors for a discussion of the various kinds of errors LIGGGHTS(R)-WSU can or can't detect, a list of all ERROR and WARNING messages, and what to do about them.

LIGGGHTS(R)-WSU 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.

LIGGGHTS(R)-WSU 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.

2.7 Command-line options

At run time, LIGGGHTS(R)-WSU recognizes several optional command-line switches which may be used in any order. Either the full word or a one-or-two letter abbreviation can be used:

- -e or -echo
- -i or -in
- -h or -help
- -l or -log
- -nc or -nocite
- -p or -partition
- -pl or -plog
- -ps or -pscreen
- -r or -restart
- -ro or -reorder
- -sc or -screen
- -sf or -suffix
- -v or -var

For example, lmp_ibm might be launched as follows:

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

Here are the details on the options:

```
-echo style
```

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

```
-in file
```

Specify a file to use as an input script. This is an optional switch when running LIGGGHTS(R)-WSU in one-partition mode. If it is not specified, LIGGGHTS(R)-WSU reads its input script from stdin - e.g. lmp_linux < in.run. This is a required switch when running LIGGGHTS(R)-WSU in multi-partition mode,

since multiple processors cannot all read from stdin.

```
-help
```

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

```
-log file
```

Specify a log file for LIGGGHTS(R)-WSU to write status information to. In one-partition mode, if the switch is not used, LIGGGHTS(R)-WSU writes to the file log.lammps. If this switch is used, LIGGGHTS(R)-WSU 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](log) command in the input script will override this setting. Option -plog will override the name of the partition log files file.N.

```
-nocite
```

Disable writing the log.cite file which is normally written to list references for specific cite-able features used during a LIGGGHTS(R)-WSU run. See the citation page for more details.

```
-partition 8x2 4 5 ...
```

Invoke LIGGGHTS(R)-WSU in multi-partition mode. When LIGGGHTS(R)-WSU is run on P processors and this switch is not used, LIGGGHTS(R)-WSU 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.

To run multiple independent simulatoins from one input script, using multiple partitions, see Section_howto 4 of the manual. World- and universe-style variables are useful in this context.

```
-plog file
```

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

```
-pscreen file
```

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

```
-restart restartfile datafile
```

Convert the restart file into a data file and immediately exit. This is the same operation as if the following 2-line input script were run:

```
read_restart restartfile
write_data datafile
```

Note that the specified restartfile and datafile can have wild-card characters ("*",%") as described by the read_restart and write_data commands. But a filename such as file.* will need to be enclosed in quotes to avoid shell expansion of the "*" character.

```
-reorder nth N
-reorder custom filename
```

Reorder the processors in the MPI communicator used to instantiate LIGGGHTS(R)-WSU, in one of several ways. The original MPI communicator ranks all P processors from 0 to P-1. The mapping of these ranks to physical processors is done by MPI before LIGGGHTS(R)-WSU begins. It may be useful in some cases to alter the rank order. E.g. to insure that cores within each node are ranked in a desired order. Or when using the run_style verlet/split command with 2 partitions to insure that a specific Kspace processor (in the 2nd partition) is matched up with a specific set of processors in the 1st partition. See the Section_accelerate doc pages for more details.

If the keyword *nth* is used with a setting *N*, then it means every Nth processor will be moved to the end of the ranking. This is useful when using the run_style verlet/split command with 2 partitions via the -partition command-line switch. The first set of processors will be in the first partition, the 2nd set in the 2nd partition. The -reorder command-line switch can alter this so that the 1st N procs in the 1st partition and one proc in the 2nd partition will be ordered consecutively, e.g. as the cores on one physical node. This can boost performance. For example, if you use "-reorder nth 4" and "-partition 9 3" and you are running on 12 processors, the processors will be reordered from

```
0 1 2 3 4 5 6 7 8 9 10 11
```

to

```
0 1 2 4 5 6 8 9 10 3 7 11
```

so that the processors in each partition will be

```
0 1 2 4 5 6 8 9 10
3 7 11
```

See the "processors" command for how to insure processors from each partition could then be grouped optimally for quad-core nodes.

If the keyword is *custom*, then a file that specifies a permutation of the processor ranks is also specified. The format of the reorder file is as follows. Any number of initial blank or comment lines (starting with a "#" character) can be present. These should be followed by P lines of the form:

```
I J
```

where P is the number of processors LIGGGHTS(R)-WSU was launched with. Note that if running in multi-partition mode (see the -partition switch above) P is the total number of processors in all partitions. The I and J values describe a permutation of the P processors. Every I and J should be values from 0 to P-1 inclusive. In the set of P I values, every proc ID should appear exactly once. Ditto for the set of P J values. A single I,J pairing means that the physical processor with rank I in the original MPI communicator will have rank J in the reordered communicator.

Note that rank ordering can also be specified by many MPI implementations, either by environment variables that specify how to order physical processors, or by config files that specify what physical processors to assign to each MPI rank. The -reorder switch simply gives you a portable way to do this without relying on MPI itself. See the processors out command for how to output info on the final assignment of physical processors to the LIGGGHTS(R)-WSU simulation domain.

```
-screen file
```

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

```
-var name value1 value2 ...
```

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

NOTE: Currently, the command-line parser looks for arguments that start with "-" to indicate new switches. Thus you cannot specify multiple variable values if any of they start with a "-", e.g. a negative numeric value. It is OK if the first value1 starts with a "-", since it is automatically skipped.

2.8 LIGGGHTS(R)-WSU screen output

As LIGGGHTS(R)-WSU 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, LIGGGHTS(R)-WSU 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, LIGGGHTS(R)-WSU prints the final thermodynamic state and a total run time for the simulation. It then appends statistics about the CPU time and storage requirements for the simulation. An example set of statistics is shown here:

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

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

Nlocal:    1002 ave, 1015 max, 989 min
Histogram: 1 0 0 0 0 0 0 0 0 1
Nghost:    8720 ave, 8724 max, 8716 min
Histogram: 1 0 0 0 0 0 0 0 0 1
```

```
Neighs:    354141 ave, 361422 max, 346860 min
Histogram: 1 0 0 0 0 0 0 0 0 1

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

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

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

The first line lists the initial and final energy, as well as the energy on the next-to-last iteration. The next 2 lines give a measure of the gradient of the energy (force on all atoms). The 2-norm is the "length" of this force vector; the inf-norm is the largest component. The last 2 lines are statistics on how many iterations and force-evaluations the minimizer required. Multiple force evaluations are typically done at each iteration to perform a 1d line minimization in the search direction.

set command

Syntax:

```
set style ID keyword values ...
```

- style = *atom* or *type* or *mol* or *group* or *region*
- ID = atom ID range or type range or mol ID range or group ID or region ID
- one or more keyword/value pairs may be appended
- keyword = *type* or *type/fraction* or *mol* or *x* or *y* or *z* or *charge* or *quat* or *quat/random* or *diameter* or *shape* or *length* or *tri* or *theta* or *angmom* or *mass* or *density* or *volume* or *image* or *bond* or *property/atom*

```
type value = atom type
    value can be an atom-style variable (see below)
  type/fraction values = type fraction seed
    type = new atom type
    fraction = fraction of selected atoms to set to new atom type
    seed = random # seed (positive integer)
  mol value = molecule ID
    value can be an atom-style variable (see below)
  x,y,z value = atom coordinate (distance units)
    value can be an atom-style variable (see below)
  charge value = atomic charge (charge units)
    value can be an atom-style variable (see below)
  quat values = a b c theta
    a,b,c = unit vector to rotate particle around via right-hand rule
    theta = rotation angle (degrees)
    any of a,b,c,theta can be an atom-style variable (see below)
  quat/random value = seed
    seed = random # seed (positive integer) for quaternion orientations
  quat_direct values = q1 q2 q3 q4
    q1,q2,q3,q4 = components of the unit quaternion, alternative to quat a b c theta
    any of q1,q2,q3,q4 can be an atom-style variable (see below)
  diameter value = diameter of spherical particle (distance units)
    value can be an atom-style variable (see below)
  shape value = Sx Sy Sz
    Sx,Sy,Sz = 3 diameters of ellipsoid or semi-axes of superquadric (distance units)
  length value = len
    len = length of line segment (distance units)
    len can be an atom-style variable (see below)
  tri value = side
    side = side length of equilateral triangle (distance units)
    side can be an atom-style variable (see below)
  theta value = angle (degrees)
    angle = orientation of line segment with respect to x-axis
    angle can be an atom-style variable (see below)
  angmom values = Lx Ly Lz
    Lx,Ly,Lz = components of angular momentum vector (distance-mass-velocity units)
    any of Lx,Ly,Lz can be an atom-style variable (see below)
  mass value = per-atom mass (mass units)
    value can be an atom-style variable (see below)
  density value = particle density for sphere or ellipsoid (mass/distance^3 or mass/distan
    value can be an atom-style variable (see below)
  volume value = particle volume for Peridynamic particle (distance^3 units)
    value can be an atom-style variable (see below)
  image nx ny nz
    nx,ny,nz = which periodic image of the simulation box the atom is in
  add value = yes no
    yes = add per-atom quantities to a region or a group
```

```
     until value = final timestep
       final timestep = the final timestep value until which the per-atom quantity is to be ac
     property/atom value = varname var_value0 var_value1 ....
       varname = name of the variable to be set
       var_value0, var_value1... = values of the property to be set.
       value can be an atom-style variable (see below)
     roundness values = n1 n2
       n1, n2 = superquadric roundness/blockiness parameters, more or equal 2
```

Examples:

```
set group solvent type 2
set group solvent type/fraction 2 0.5 12393
set group edge bond 4
set region half charge 0.5
set type 3 charge 0.5
set type 1*3 charge 0.5
set atom 100*200 x 0.5 y 1.0
set atom 1492 type 3
set property/atom Temp 273.15
set atom 3 type 2 shape 0.001 0.001 0.001 roundness 10.0 10.0 density 2500
```

Description:

Set one or more properties of one or more atoms. Since atom properties are initially assigned by the read_data, read_restart or create_atoms commands, this command changes those assignments. This can be useful for overriding the default values assigned by the create_atoms command (e.g. charge = 0.0). It can be useful for altering pairwise and molecular force interactions, since force-field coefficients are defined in terms of types. It can be used to change the labeling of atoms by atom type or molecule ID when they are output in dump files. It can also be useful for debugging purposes; i.e. positioning an atom at a precise location to compute subsequent forces or energy.

Note that the *style* and *ID* arguments determine which atoms have their properties reset. The remaining keywords specify which properties to reset and what the new values are. Some strings like *type* or *mol* can be used as a style and/or a keyword.

This section describes how to select which atoms to change the properties of, via the *style* and *ID* arguments.

The style *atom* selects all the atoms in a range of atom IDs. The style *type* selects all the atoms in a range of types. The style *mol* selects all the atoms in a range of molecule IDs.

In each of the range cases, the range can be specified as a single numeric value, or a wildcard asterisk can be used to specify a range of values. This takes the form "*" or "*n" or "n*" or "m*n". For example, for the style *type*, if N = the number of atom types, then an asterisk with no numeric values means all types from 1 to N. A leading asterisk means all types from 1 to n (inclusive). A trailing asterisk means all types from n to N (inclusive). A middle asterisk means all types from m to n (inclusive). For all the styles except *mol*, the lowest value for the wildcard is 1; for *mol* it is 0.

The style *group* selects all the atoms in the specified group. The style *region* selects all the atoms in the specified geometric region. See the group and region commands for details of how to specify a group or region.

This section describes the keyword options for which properties to change, for the selected atoms.

Note that except where explicitly prohibited below, all of the keywords allow an atom-style variable to be used as the specified value(s). If the value is a variable, it should be specified as v_name, where name is the variable name. In this case, the variable will be evaluated, and its resulting per-atom value used to determine

the value assigned to each selected atom.

Atom-style variables can specify formulas with various mathematical functions, and include thermo_style command keywords for the simulation box parameters and timestep and elapsed time. They can also include per-atom values, such as atom coordinates. Thus it is easy to specify a time-dependent or spatially-dependent set of per-atom values. As explained on the variable doc page, atomfile-style variables can be used in place of atom-style variables, and thus as arguments to the set command. Atomfile-style variables read their per-atoms values from a file.

IMPORTANT NOTE: Atom-style and atomfile-style variables return floating point per-atom values. If the values are assigned to an integer variable, such as the molecule ID, then the floating point value is truncated to its integer portion, e.g. a value of 2.6 would become 2.

Keyword *type* sets the atom type for all selected atoms. The specified value must be from 1 to ntypes, where ntypes was set by the create_box command or the *atom types* field in the header of the data file read by the read_data command.

Keyword *type/fraction* sets the atom type for a fraction of the selected atoms. The actual number of atoms changed is not guaranteed to be exactly the requested fraction, but should be statistically close. Random numbers are used in such a way that a particular atom is changed or not changed, regardless of how many processors are being used. This keyword does not allow use of an atom-style variable.

Keyword *mol* sets the molecule ID for all selected atoms. The atom style being used must support the use of molecule IDs.

Keywords *x*, *y*, *z*, and *charge* set the coordinates or charge of all selected atoms. For *charge*, the atom style being used must support the use of atomic charge.

Keyword *dipole* uses the specified x,y,z values as components of a vector to set as the orientation of the dipole moment vectors of the selected atoms. The magnitude of the dipole moment is set by the length of this orientation vector.

Keyword *dipole/random* randomizes the orientation of the dipole moment vectors of the selected atoms and sets the magnitude of each to the specified *Dlen* value. For 2d systems, the z component of the orientation is set to 0.0. Random numbers are used in such a way that the orientation of a particular atom is the same, regardless of how many processors are being used. This keyword does not allow use of an atom-style variable.

Keyword *quat* uses the specified values to create a quaternion (4-vector) that represents the orientation of the selected atoms. The particles must be ellipsoids as defined by the atom_style ellipsoid command, triangles as defined by the atom_style tri command or superquadric as defined by the atom_style tri command. Note that particles defined by atom_style ellipsoid have 3 shape parameters. The 3 values must be non-zero for each particle set by this command. They are used to specify the aspect ratios of an ellipsoidal particle, which is oriented by default with its x-axis along the simulation box's x-axis, and similarly for y and z. If this body is rotated (via the right-hand rule) by an angle theta around a unit rotation vector (a,b,c), then the quaternion that represents its new orientation is given by (cos(theta/2), a*sin(theta/2), b*sin(theta/2), c*sin(theta/2)). The theta and a,b,c values are the arguments to the *quat* keyword. LIGGGHTS(R)-WSU normalizes the quaternion in case (a,b,c) was not specified as a unit vector. For 2d systems, the a,b,c values are ignored, since a rotation vector of (0,0,1) is the only valid choice.

Keyword *quat/random* randomizes the orientation of the quaternion of the selected atoms. The particles must be ellipsoids as defined by the atom_style ellipsoid command, triangles as defined by the atom_style tri command or superquadric as defined by the atom_style superquadric command. Random numbers are used in such a way that the orientation of a particular atom is the same, regardless of how many processors are being used. For 2d systems, only orientations in the xy plane are generated. As with keyword *quat*, for ellipsoidal

set command

and superquadric particles, the 3 shape values must be non-zero for each particle set by this command. This keyword does not allow use of an atom-style variable.

Keyword *quat_direct* simply creates a quaternion from it's values instead of calculting these from a unit rotation vector and rotation angle. These keyword is an alternative to the keyword *quat*.

Keyword *diameter* sets the size of the selected atoms. The particles must be finite-size spheres as defined by the atom_style sphere command. The diameter of a particle can be set to 0.0, which means they will be treated as point particles. Note that this command does not adjust the particle mass, even if it was defined with a density, e.g. via the read_data command.

Keyword *shape* sets the size and shape of the selected atoms. The particles must be ellipsoids or superquadric as defined by the atom_style ellipsoid or atom_style superquadric command. The *Sx*, *Sy*, *Sz* settings are the 3 diameters of the ellipsoid or the 3 semi-axes lenghts of the superquadric in each direction. All 3 can be set to the same value, which means the ellipsoid is effectively a sphere. They can also all be set to 0.0 which means the particle will be treated as a point particle. Note that this command does not adjust the particle mass, even if it was defined with a density, e.g. via the read_data command.

Keyword *length* sets the length of selected atoms. The particles must be line segments as defined by the atom_style line command. If the specified value is non-zero the line segment is (re)set to a length = the specified value, centered around the particle position, with an orientation along the x-axis. If the specified value is 0.0, the particle will become a point particle. Note that this command does not adjust the particle mass, even if it was defined with a density, e.g. via the read_data command.

Keyword *tri* sets the size of selected atoms. The particles must be triangles as defined by the atom_style tri command. If the specified value is non-zero the triangle is (re)set to be an equilateral triangle in the xy plane with side length = the specified value, with a centroid at the particle position, with its base parallel to the x axis, and the y-axis running from the center of the base to the top point of the triangle. If the specified value is 0.0, the particle will become a point particle. Note that this command does not adjust the particle mass, even if it was defined with a density, e.g. via the read_data command.

Keyword *theta* sets the orientation of selected atoms. The particles must be line segments as defined by the atom_style line command. The specified value is used to set the orientation angle of the line segments with respect to the x axis.

Keyword *angmom* sets the angular momentum of selected atoms. The particles must be ellipsoids as defined by the atom_style ellipsoid command, triangles as defined by the atom_style tri command or superquadrics as defined by the atom_style superquadric command. The angular momentum vector of the particles is set to the 3 specified components.

Keyword *mass* sets the mass of all selected particles. The particles must have a per-atom mass attribute, as defined by the atom_style command. See the "mass" command for how to set mass values on a per-type basis.

Keyword *density* also sets the mass of all selected particles, but in a different way. The particles must have a per-atom mass attribute, as defined by the atom_style command. If the atom has a radius attribute (see atom_style sphere) and its radius is non-zero, its mass is set from the density and particle volume. If the atom has a shape attribute (see atom_style ellipsoid) and its 3 shape parameters are non-zero, then its mass is set from the density and particle volume. If the atom has a length attribute (see atom_style line) and its length is non-zero, then its mass is set from the density and line segment length (the input density is assumed to be in mass/distance units). If the atom has an area attribute (see atom_style tri) and its area is non-zero, then its mass is set from the density and triangle area (the input density is assumed to be in mass/distance^2 units). If none of these cases are valid, then the mass is set to the density value directly (the input density is assumed to be in mass units).

set command   594

Keyword *volume* sets the volume of all selected particles. Currently, only the atom_style peri command defines particles with a volume attribute. Note that this command does not adjust the particle mass.

Keyword *image* sets which image of the simulation box the atom is considered to be in. An image of 0 means it is inside the box as defined. A value of 2 means add 2 box lengths to get the true value. A value of -1 means subtract 1 box length to get the true value. LIGGGHTS(R)-WSU updates these flags as atoms cross periodic boundaries during the simulation. The flags can be output with atom snapshots via the dump command. If a value of NULL is specified for any of nx,ny,nz, then the current image value for that dimension is unchanged. For non-periodic dimensions only a value of 0 can be specified. This keyword does not allow use of atom-style variables.

This command can be useful after a system has been equilibrated and atoms have diffused one or more box lengths in various directions. This command can then reset the image values for atoms so that they are effectively inside the simulation box, e.g if a diffusion coefficient is about to be measured via the compute msd command. Care should be taken not to reset the image flags of two atoms in a bond to the same value if the bond straddles a periodic boundary (rather they should be different by +/- 1). This will not affect the dynamics of a simulation, but may mess up analysis of the trajectories if a LIGGGHTS(R)-WSU diagnostic or your own analysis relies on the image flags to unwrap a molecule which straddles the periodic box.

Keywords *bond* set the bond type of all bonds of selected atoms to the specified value from 1 to nbondtypes. All atoms in a particular bond must be selected atoms in order for the change to be made. The value of nbondtype was set by the *bond types* field in the header of the data file read by the read_data command. These keywords do not allow use of an atom-style variable.

Keyword *property/atom* can update per-particle properties defined by a fix property/atom command. varname is the name of the variable you want to set, it is followed by the values that should be assigned to this variable. The number of values provided as var_value**i** must correspond to the number of values needed for the update, i.e. only var_value**0** if the defined property is a scalar, and the appropriate number if the property is a vector. See fix property/atom for details on how to define such a per-particle property.

Keyword *add* and *until* are used to add a per-atom quantity (or property) in addition to the keyword *property/atom*. The *add* keyword is used to turn on the addition of a quantity in a region or a group until the timestep defined by the *until* keyword.

Keywords *shape* and *roundness* define superquadric shape of a particle.

Restrictions:

You cannot set an atom attribute (e.g. *mol* or *q* or *volume*) if the atom_style does not have that attribute.

This command requires inter-processor communication to coordinate the setting of bond types (angle types, etc). This means that your system must be ready to perform a simulation before using one of these keywords (force fields set, atom mass set, etc). This is not necessary for other keywords.

Using the *region* style with the bond (angle, etc) keywords can give unpredictable results if there are bonds (angles, etc) that straddle periodic boundaries. This is because the region may only extend up to the boundary and partner atoms in the bond (angle, etc) may have coordinates outside the simulation box if they are ghost atoms.

Keywords *quat_direct* and *roundness* require atom_style *superquadric*

Related commands:

create_box, create_atoms, read_data

set command   595

Default: none

shell command

Syntax:

```
shell cmd args
```

- cmd = *cd* or *mkdir* or *mv* or *rm* or *rmdir* or *putenv* or arbitrary command

```
  cd arg = dir
     dir = directory to change to
   mkdir args = dir1 dir2 ...
     dir1,dir2 = one or more directories to create
   mv args = old new
     old = old filename
     new = new filename
   rm args = file1 file2 ...
     file1,file2 = one or more filenames to delete
   rmdir args = dir1 dir2 ...
     dir1,dir2 = one or more directories to delete
   putenv args = var1=value1 var2=value2
     var=value = one of more definitions of environment variables
   anything else is passed as a command to the shell for direct execution
```

Examples:

```
shell cd sub1
shell cd ..
shell mkdir tmp1 tmp2 tmp3
shell rmdir tmp1
shell mv log.lammps hold/log.1
shell rm TMP/file1 TMP/file2
shell my_setup file1 10 file2
shell my_post_process 100 dump.out
```

Description:

Execute a shell command. A few simple file-based shell commands are supported directly, in Unix-style syntax. Any command not listed above is passed as-is to the C-library system() call, which invokes the command in a shell.

This is means to invoke other commands from your input script. For example, you can move files around in preparation for the next section of the input script. Or you can run a program that pre-processes data for input into LIGGGHTS(R)-WSU. Or you can run a program that post-processes LIGGGHTS(R)-WSU output data.

With the exception of *cd*, all commands, including ones invoked via a system() call, are executed by only a single processor, so that files/directories are not being manipulated by multiple processors.

The *cd* cmd executes the Unix "cd" command to change the working directory. All subsequent LIGGGHTS(R)-WSU commands that read/write files will use the new directory. All processors execute this command.

The *mkdir* cmd executes the Unix "mkdir" command to create one or more directories.

The *mv* cmd executes the Unix "mv" command to rename a file and/or move it to a new directory.

The *rm* cmd executes the Unix "rm" command to remove one or more files.

The *rmdir* cmd executes the Unix "rmdir" command to remove one or more directories. A directory must be empty to be successfully removed.

The *putenv* cmd defines or updates an environment variable directly. Since this command does not pass through the shell, no shell variable expansion or globbing is performed, only the usual substitution for LIGGGHTS(R)-WSU variables defined with the [variable](#) command is performed. The resulting string is then used literally.

Any other cmd is passed as-is to the shell along with its arguments as one string, invoked by the C-library system() call. For example, these lines in your input script:

```
variable n equal 10
variable foo string file2
shell my_setup file1 $n ${foo}
```

would be the same as invoking

```
% my_setup file1 10 file2
```

from a command-line prompt. The executable program "my_setup" is run with 3 arguments: file1 10 file2.

Restrictions:

LIGGGHTS(R)-WSU does not detect errors or print warnings when any of these commands execute. E.g. if the specified directory does not exist, executing the *cd* command will silently do nothing.

Related commands: none

Default: none

thermo command

Syntax:

```
thermo N
```

- N = output thermodynamics every N timesteps
- N can be a variable (see below)

Examples:

```
thermo 100
```

Description:

Compute and print thermodynamic info (e.g. temperature, energy, pressure) on timesteps that are a multiple of N and at the beginning and end of a simulation. A value of 0 will only print thermodynamics at the beginning and end.

The content and format of what is printed is controlled by the thermo_style and thermo_modify commands.

Instead of a numeric value, N can be specifed as an equal-style variable, which should be specified as v_name, where name is the variable name. In this case, the variable is evaluated at the beginning of a run to determine the next timestep at which thermodynamic info will be written out. On that timestep, the variable will be evaluated again to determine the next timestep, etc. Thus the variable should return timestep values. See the stagger() and logfreq() and stride() math functions for equal-style variables, as examples of useful functions to use in this context. Other similar math functions could easily be added as options for equal-style variables.

For example, the following commands will output thermodynamic info at timesteps 0,10,20,30,100,200,300,1000,2000,etc:

```
variable        s equal logfreq(10,3,10)
thermo          v_s
```

Restrictions: none

Related commands:

thermo_style, thermo_modify

Default:

```
thermo 0
```

thermo_modify command

Syntax:

```
thermo_modify keyword value ...
```

- one or more keyword/value pairs may be listed
- keyword = *lost* or *norm* or *flush* or *line* or *format*

```
  lost value = error or warn or ignore
    norm value = yes or no
    flush value = yes or no
    line value = one or multi
    format values = int string or float string or M string
      M = integer from 1 to N, where N = # of quantities being printed
      string = C-style format string
```

Examples:

```
thermo_modify lost ignore flush yes
thermo_modify line multi format float %g
```

Description:

Set options for how thermodynamic information is computed and printed by LIGGGHTS(R)-WSU.

IMPORTANT NOTE: These options apply to the currently defined thermo style. When you specify a [thermo_style](#) command, all thermodynamic settings are restored to their default values, including those previously reset by a thermo_modify command. Thus if your input script specifies a thermo_style command, you should use the thermo_modify command after it.

The *lost* keyword determines whether LIGGGHTS(R)-WSU checks for lost atoms each time it computes thermodynamics and what it does if atoms are lost. An atom can be "lost" if it moves across a non-periodic simulation box [boundary](#) or if it moves more than a box length outside the simulation domain (or more than a processor sub-domain length) before reneighboring occurs. The latter case is typically due to bad dynamics, e.g. too large a timestep or huge forces and velocities. If the value is *ignore*, LIGGGHTS(R)-WSU does not check for lost atoms. If the value is *error* or *warn*, LIGGGHTS(R)-WSU checks and either issues an error or warning. The code will exit with an error and continue with a warning. A warning will only be issued once, the first time an atom is lost. This can be a useful debugging option.

IMPORTANT NOTE: For compuatational efficiency, this *lost* check is just a simple check if the total number of atoms in the system decreases. So for simulations where particles are continuously inserted, "lost" atoms might not be detected.

The *norm* keyword determines whether various thermodynamic output values are normalized by the number of atoms or not, depending on whether it is set to *yes* or *no*. Different unit styles have different defaults for this setting (see below). Even if *norm* is set to *yes*, a value is only normalized if it is an "extensive" quantity, meaning that it scales with the number of atoms in the system. For the thermo keywords described by the doc page for the [thermo_style](#) command, all energy-related keywords are extensive. Other keywords such are "intensive" meaning their value is independent (in a statistical sense) of the number of atoms in the system and thus are never normalized. For thermodynamic output values extracted from fixes and computes in a [thermo_style custom](#) command, the doc page for the individual [fix](#) or [compute](#) lists whether the value is "extensive" or "intensive" and thus whether it is normalized. Thermodynamic output values calculated by a

variable formula are assumed to be "intensive" and thus are never normalized. You can always include a divide by the number of atoms in the variable formula if this is not the case.

The *flush* keyword invokes a flush operation after thermodynamic info is written to the log file. This insures the output in that file is current (no buffering by the OS), even if LIGGGHTS(R)-WSU halts before the simulation completes.

The *line* keyword determines whether thermodynamics will be printed as a series of numeric values on one line or in a multi-line format with 3 quantities with text strings per line and a dashed-line header containing the timestep and CPU time. This modify option overrides the *one* and *multi* thermo_style settings.

The *format* keyword sets the numeric format of individual printed quantities. The *int* and *float* keywords set the format for all integer or floating-point quantities printed. The setting with a numeric value M (e.g. format 5 %10.4g) sets the format of the Mth value printed in each output line, e.g. the 5th column of output in this case. If the format for a specific column has been set, it will take precedent over the *int* or *float* setting.

IMPORTANT NOTE: The thermo output values *step* and *atoms* are stored internally as 8-byte signed integers, rather than the usual 4-byte signed integers. When specifying the "format int" keyword you can use a "%d"-style format identifier in the format string and LIGGGHTS(R)-WSU will convert this to the corresponding "%lu" form when it is applied to those keywords. However, when specifying the "format M string" keyword for *step* and *natoms*, you should specify a string appropriate for an 8-byte signed integer, e.g. one with "%ld".

Restrictions: none

Related commands:

thermo, thermo_style

Default:

The option defaults are lost = ignore, norm = yes for unit style of *lj*, norm = no for unit style of *real* and *metal*, flush = no.

The defaults for the line and format options depend on the thermo style. For styles "one" and "custom", the line and format defaults are "one", "%8d", and "%12.8g". For style "multi", the line and format defaults are "multi", "%8d", and "%14.4f".

thermo_style command

Syntax:

```
thermo_style style args
```

- style = *one* or *multi* or *custom*
- args = list of arguments for a particular style

```
one args = none
  multi args = none
  custom args = list of attributes
    possible attributes = step, elapsed, elaplong, dt, time,
                          cpu, tpcpu, spcpu, cpuremain, part, cu
                          atoms, ke,
                          vol, lx, ly, lz, xlo, xhi, ylo, yhi, zlo, zhi,
                          xy, xz, yz, xlat, ylat, zlat,
                          pxx, pyy, pzz, pxy, pxz, pyz,
                          fmax, fnorm,
                          cella, cellb, cellc, cellalpha, cellbeta, cellgamma,
                          c_ID, c_ID[I], c_ID[I][J],
                          f_ID, f_ID[I], f_ID[I][J],
                          v_name
      step = timestep
      elapsed = timesteps since start of this run
      elaplong = timesteps since start of initial run in a series of runs
      dt = timestep size
      time = simulation time
      cpu = elapsed CPU time in seconds
      tpcpu = time per CPU second
      spcpu = timesteps per CPU second
      cpuremain = estimated CPU time remaining in run
      part = which partition (0 to Npartition-1) this is
      cu = timesteps per CPU second
      atoms = # of atoms
      vol = volume
      lx,ly,lz = box lengths in x,y,z
      xlo,xhi,ylo,yhi,zlo,zhi = box boundaries
      xy,xz,yz = box tilt for triclinic (non-orthogonal) simulation boxes
      xlat,ylat,zlat = lattice spacings as calculated by lattice command
      pxx,pyy,pzz,pxy,pxz,pyz = 6 components of pressure tensor
      fmax = max component of force on any atom in any dimension
      fnorm = length of force vector for all atoms
      cella,cellb,cellc = periodic cell lattice constants a,b,c
      cellalpha, cellbeta, cellgamma = periodic cell angles alpha,beta,gamma
      c_ID = global scalar value calculated by a compute with ID
      c_ID[I] = Ith component of global vector calculated by a compute with ID
      c_ID[I][J] = I,J component of global array calculated by a compute with ID
      f_ID = global scalar value calculated by a fix with ID
      f_ID[I] = Ith component of global vector calculated by a fix with ID
      f_ID[I][J] = I,J component of global array calculated by a fix with ID
      v_name = scalar value calculated by an equal-style variable with name
```

Examples:

```
thermo_style one
thermo_style custom step v_abc
```

Description:

Set the style and content for printing thermodynamic data to the screen and log file.

Style *one* prints a one-line summary of thermodynamic info that is the equivalent of "thermo_style custom step atoms ke cpu". The line contains only numeric values.

Style *multi* prints a multiple-line listing of thermodynamic info that is the equivalent of "thermo_style custom step atoms ke cpu". The listing contains numeric values and a string ID for each quantity.

Style *custom* is the most general setting and allows you to specify which of the keywords listed above you want printed on each thermodynamic timestep. Note that the keywords c_ID, f_ID, v_name are references to computes, fixes, and equal-style variables that have been defined elsewhere in the input script or can even be new styles which users have added to LIGGGHTS(R)-WSU (see the Section_modify section of the documentation). Thus the *custom* style provides a flexible means of outputting essentially any desired quantity as a simulation proceeds.

All styles except *custom* have *vol* appended to their list of outputs if the simulation box volume changes during the simulation.

The values printed by the various keywords are instantaneous values, calculated on the current timestep. Time-averaged quantities, which include values from previous timesteps, can be output by using the f_ID keyword and accessing a fix that does time-averaging such as the fix ave/time command.

Options invoked by the thermo_modify command can be used to set the one- or multi-line format of the print-out, the normalization of thermodynamic output (total values versus per-atom values for extensive quantities (ones which scale with the number of atoms in the system), and the numeric precision of each printed value.

IMPORTANT NOTE: When you use a "thermo_style" command, all thermodynamic settings are restored to their default values, including those previously set by a thermo_modify command. Thus if your input script specifies a thermo_style command, you should use the thermo_modify command after it.

The *step*, *elapsed*, and *elaplong* keywords refer to timestep count. *Step* is the current timestep, or iteration count when a minimization is being performed. *Elapsed* is the number of timesteps elapsed since the beginning of this run. *Elaplong* is the number of timesteps elapsed since the beginning of an initial run in a series of runs. See the *start* and *stop* keywords for the run for info on how to invoke a series of runs that keep track of an initial starting time. If these keywords are not used, then *elapsed* and *elaplong* are the same value.

The *dt* keyword is the current timestep size in time units. The *time* keyword is the current elapsed simulation time, also in time units, which is simply (step*dt) if the timestep size has not changed and the timestep has not been reset. If the timestep has changed (e.g. via fix dt/reset) or the timestep has been reset (e.g. via the "reset_timestep" command), then the simulation time is effectively a cummulative value up to the current point.

The *cpu* keyword is elapsed CPU seconds since the beginning of this run. The *tpcpu* and *spcpu* keywords are measures of how fast your simulation is currently running. The *tpcpu* keyword is simulation time per CPU second, where simulation time is in time units. E.g. for metal units, the *tpcpu* value would be picoseconds per CPU second. The *spcpu* keyword is the number of timesteps per CPU second. Both quantities are on-the-fly metrics, measured relative to the last time they were invoked. Thus if you are printing out thermodyamic output every 100 timesteps, the two keywords will continually output the time and timestep rate for the last 100 steps. The *tpcpu* keyword does not attempt to track any changes in timestep size, e.g. due to using the fix dt/reset command.

The *cpuremain* keyword estimates the CPU time remaining in the current run, based on the time elapsed thus far. It will only be a good estimate if the CPU time/timestep for the rest of the run is similar to the preceding

thermo_style command

timesteps. On the initial timestep the value will be 0.0 since there is no history to estimate from. For a minimization run performed by the "minimize" command, the estimate is based on the *maxiter* parameter, assuming the minimization will proceed for the maximum number of allowed iterations.

The *part* keyword is useful for multi-replica or multi-partition simulations to indicate which partition this output and this file corresponds to, or for use in a variable to append to a filename for output specific to this partition. See Section_start 7 of the manual for details on running in multi-partition mode.

The *fmax* and *fnorm* keywords are useful for monitoring the progress of an energy minimization. The *fmax* keyword calculates the maximum force in any dimension on any atom in the system, or the infinity-norm of the force vector for the system. The *fnorm* keyword calculates the 2-norm or length of the force vector.

The keywords *cella*, *cellb*, *cellc*, *cellalpha*, *cellbeta*, *cellgamma*, correspond to the usual crystallographic quantities that define the periodic unit cell of a crystal. See this section of the doc pages for a geometric description of triclinic periodic cells, including a precise defintion of these quantities in terms of the internal LIGGGHTS(R)-WSU cell dimensions *lx*, *ly*, *lz*, *yz*, *xz*, *xy*.

The *c_ID* and *c_ID[I]* and *c_ID[I][J]* keywords allow global values calculated by a compute to be output. As discussed on the compute doc page, computes can calculate global, per-atom, or local values. Only global values can be referenced by this command. However, per-atom compute values can be referenced in a variable and the variable referenced by thermo_style custom, as discussed below.

The ID in the keyword should be replaced by the actual ID of a compute that has been defined elsewhere in the input script. See the compute command for details. If the compute calculates a global scalar, vector, or array, then the keyword formats with 0, 1, or 2 brackets will reference a scalar value from the compute.

Note that some computes calculate "intensive" global quantities like temperature; others calculate "extensive" global quantities like kinetic energy that are summed over all atoms in the compute group. Intensive quantities are printed directly without normalization by thermo_style custom. Extensive quantities may be normalized by the total number of atoms in the simulation (NOT the number of atoms in the compute group) when output, depending on the thermo_modify norm option being used.

The *f_ID* and *f_ID[I]* and *f_ID[I][J]* keywords allow global values calculated by a fix to be output. As discussed on the fix doc page, fixes can calculate global, per-atom, or local values. Only global values can be referenced by this command. However, per-atom fix values can be referenced in a variable and the variable referenced by thermo_style custom, as discussed below.

The ID in the keyword should be replaced by the actual ID of a fix that has been defined elsewhere in the input script. See the fix command for details. If the fix calculates a global scalar, vector, or array, then the keyword formats with 0, 1, or 2 brackets will reference a scalar value from the fix.

Note that some fixes calculate "intensive" global quantities like timestep size; others calculate "extensive" global quantities like energy that are summed over all atoms in the fix group. Intensive quantities are printed directly without normalization by thermo_style custom. Extensive quantities may be normalized by the total number of atoms in the simulation (NOT the number of atoms in the fix group) when output, depending on the thermo_modify norm option being used.

The *v_name* keyword allow the current value of a variable to be output. The name in the keyword should be replaced by the variable name that has been defined elsewhere in the input script. Only equal-style variables can be referenced. See the variable command for details. Variables of style *equal* can reference per-atom properties or thermodynamic keywords, or they can invoke other computes, fixes, or variables when evaluated, so this is a very general means of creating thermodynamic output.

Note that equal-style variables are assumed to be "intensive" global quantities, which are thus printed as-is,

without normalization by thermo_style custom. You can include a division by "natoms" in the variable formula if this is not the case.

Restrictions:

This command must come after the simulation box is defined by a read_data, read_restart, or create_box command.

Related commands:

thermo, thermo_modify, fix_modify,

Default:

```
thermo_style one
```

timestep command

Syntax:

```
timestep dt
```

- dt = timestep size (time units)

Examples:

```
timestep 2.0
timestep 0.003
```

Description:

Set the timestep size for subsequent molecular dynamics simulations. See the units command for a discussion of time units. The default value for the timestep also depends on the choice of units for the simulation; see the default values below.

When the run style is *respa*, dt is the timestep for the outer loop (largest) timestep.

Restrictions: none

Related commands:

fix dt/reset, run, run_style respa, units

Default:

timestep = 0.005 tau for units = lj
timestep = 1.0 fmsec for units = real
timestep = 0.001 psec for units = metal
timestep = 1.0e-8 sec (10 nsec) for units = si or cgs

uncompute command

Syntax:

```
uncompute compute-ID
```

- compute-ID = ID of a previously defined compute

Examples:

```
uncompute 2
uncompute lower-boundary
```

Description:

Delete a compute that was previously defined with a [compute](#) command. This also wipes out any additional changes made to the compute via the [compute_modify](#) command.

Restrictions: none

Related commands:

[compute](#)

Default: none

undump command

Syntax:

```
undump dump-ID
```

- dump-ID = ID of previously defined dump

Examples:

```
undump mine
undump 2
```

Description:

Turn off a previously defined dump so that it is no longer active. This closes the file associated with the dump.

Restrictions: none

Related commands:

dump

Default: none

unfix command

Syntax:

```
unfix fix-ID
```

- fix-ID = ID of a previously defined fix

Examples:

```
unfix 2
unfix lower-boundary
```

Description:

Delete a fix that was previously defined with a fix command. This also wipes out any additional changes made to the fix via the fix_modify command.

Restrictions: none

Related commands:

fix

Default: none

units command

Syntax:

```
units style
```

- style = *lj* or *real* or *metal* or *si* or *cgs* or *electron* or *micro* or *nano*

Examples:

```
units metal
units lj
```

Description:

This command sets the style of units used for a simulation. It determines the units of all quantities specified in the input script and data file, as well as quantities output to the screen, log file, and dump files. Typically, this command is used at the very beginning of an input script.

For all units except *lj*, LIGGGHTS(R)-WSU uses physical constants from www.physics.nist.gov. For the definition of Kcal in real units, LIGGGHTS(R)-WSU uses the thermochemical calorie = 4.184 J.

For style *lj*, all quantities are unitless. Without loss of generality, LIGGGHTS(R)-WSU sets the fundamental quantities mass, sigma, epsilon, and the Boltzmann constant = 1. The masses, distances, energies you specify are multiples of these fundamental values. The formulas relating the reduced or unitless quantity (with an asterisk) to the same quantity with units is also given. Thus you can use the mass & sigma & epsilon values for a specific material and convert the results from a unitless LJ simulation into physical quantities.

- mass = mass or m
- distance = sigma, where $x^* = x / sigma$
- time = tau, where tau = $t^* = t$ (epsilon / m / sigma^2)^1/2
- energy = epsilon, where $E^* = E / epsilon$
- velocity = sigma/tau, where $v^* = v$ tau / sigma
- force = epsilon/sigma, where $f^* = f$ sigma / epsilon
- torque = epsilon, where $t^* = t / epsilon$
- temperature = reduced LJ temperature, where $T^* = T$ Kb / epsilon
- pressure = reduced LJ pressure, where $P^* = P$ sigma^3 / epsilon
- dynamic viscosity = reduced LJ viscosity, where eta* = eta sigma^3 / epsilon / tau
- charge = reduced LJ charge, where $q^* = q$ / (4 pi perm0 sigma epsilon)^1/2
- dipole = reduced LJ dipole, moment where *mu = mu / (4 pi perm0 sigma^3 epsilon)^1/2
- electric field = force/charge, where $E^* = E$ (4 pi perm0 sigma epsilon)^1/2 sigma / epsilon
- density = mass/volume, where rho* = rho sigma^dim

Note that for LJ units, the default mode of thermodyamic output via the [thermo_style](#) command is to normalize energies by the number of atoms, i.e. energy/atom. This can be changed via the [thermo_modify norm](#) command.

For style *real*, these are the units:

- mass = grams/mole
- distance = Angstroms
- time = femtoseconds

- energy = Kcal/mole
- velocity = Angstroms/femtosecond
- force = Kcal/mole-Angstrom
- torque = Kcal/mole
- temperature = Kelvin
- pressure = atmospheres
- dynamic viscosity = Poise
- charge = multiple of electron charge (+1.0 is a proton)
- dipole = charge*Angstroms
- electric field = volts/Angstrom
- density = gram/cm^dim

For style *metal*, these are the units:

- mass = grams/mole
- distance = Angstroms
- time = picoseconds
- energy = eV
- velocity = Angstroms/picosecond
- force = eV/Angstrom
- torque = eV
- temperature = Kelvin
- pressure = bars
- dynamic viscosity = Poise
- charge = multiple of electron charge (+1.0 is a proton)
- dipole = charge*Angstroms
- electric field = volts/Angstrom
- density = gram/cm^dim

For style *si*, these are the units:

- mass = kilograms
- distance = meters
- time = seconds
- energy = Joules
- velocity = meters/second
- force = Newtons
- torque = Newton-meters
- temperature = Kelvin
- pressure = Pascals
- dynamic viscosity = Pascal*second
- charge = Coulombs
- dipole = Coulombs*meters
- electric field = volts/meter
- density = kilograms/meter^dim

For style *cgs*, these are the units:

- mass = grams
- distance = centimeters
- time = seconds
- energy = ergs
- velocity = centimeters/second
- force = dynes

- torque = dyne-centimeters
- temperature = Kelvin
- pressure = dyne/cm^2 or barye = 1.0e-6 bars
- dynamic viscosity = Poise
- charge = statcoulombs or esu
- dipole = statcoul-cm = 10^18 debye
- electric field = statvolt/cm or dyne/esu
- density = grams/cm^dim

For style *electron*, these are the units:

- mass = atomic mass units
- distance = Bohr
- time = femtoseconds
- energy = Hartrees
- velocity = Bohr/atomic time units [1.03275e-15 seconds]
- force = Hartrees/Bohr
- temperature = Kelvin
- pressure = Pascals
- charge = multiple of electron charge (+1.0 is a proton)
- dipole moment = Debye
- electric field = volts/cm

For style *micro*, these are the units:

- mass = picograms
- distance = micrometers
- time = microseconds
- energy = picogram-micrometer^2/microsecond^2
- velocity = micrometers/microsecond
- force = picogram-micrometer/microsecond^2
- torque = picogram-micrometer^2/microsecond^2
- temperature = Kelvin
- pressure = picogram/(micrometer-microsecond^2)
- dynamic viscosity = picogram/(micrometer-microsecond)
- charge = picocoulombs
- dipole = picocoulomb-micrometer
- electric field = volt/micrometer
- density = picograms/micrometer^dim

For style *nano*, these are the units:

- mass = attograms
- distance = nanometers
- time = nanoseconds
- energy = attogram-nanometer^2/nanosecond^2
- velocity = nanometers/nanosecond
- force = attogram-nanometer/nanosecond^2
- torque = attogram-nanometer^2/nanosecond^2
- temperature = Kelvin
- pressure = attogram/(nanometer-nanosecond^2)
- dynamic viscosity = attogram/(nanometer-nanosecond)
- charge = multiple of electron charge (+1.0 is a proton)
- dipole = charge-nanometer

- electric field = volt/nanometer
- density = attograms/nanometer^dim

The units command also sets the timestep size and neighbor skin distance to default values for each style:

- For style *lj* these are dt = 0.005 tau and skin = 0.3 sigma.
- For style *real* these are dt = 1.0 fmsec and skin = 2.0 Angstroms.
- For style *metal* these are dt = 0.001 psec and skin = 2.0 Angstroms.
- For style *si* these are dt = 1.0e-8 sec and skin = 0.001 meters.
- For style *cgs* these are dt = 1.0e-8 sec and skin = 0.1 cm.
- For style *electron* these are dt = 0.001 fmsec and skin = 2.0 Bohr.
- For style *micro* these are dt = 2.0 microsec and skin = 0.1 micrometers.
- For style *nano* these are dt = 0.00045 nanosec and skin = 0.1 nanometers.

Restrictions:

This command cannot be used after the simulation box is defined by a read_data or create_box command.

Related commands: none

Default:

```
units lj
```

---

variable command

Syntax:

```
variable name style args ...
```

- name = name of variable to define
- style = *delete* or *index* or *loop* or *world* or *universe* or *uloop* or *string* or *getenv* or *file* or *atomfile* or *equal* or *atom*

```
delete = no args
  index args = one or more strings
  loop args = N
    N = integer size of loop, loop from 1 to N inclusive
  loop args = N pad
    N = integer size of loop, loop from 1 to N inclusive
    pad = all values will be same length, e.g. 001, 002, ..., 100
  loop args = N1 N2
    N1,N2 = loop from N1 to N2 inclusive
  loop args = N1 N2 pad
    N1,N2 = loop from N1 to N2 inclusive
    pad = all values will be same length, e.g. 050, 051, ..., 100
  world args = one string for each partition of processors
  universe args = one or more strings
  uloop args = N
    N = integer size of loop
  uloop args = N pad
    N = integer size of loop
    pad = all values will be same length, e.g. 001, 002, ..., 100
  string arg = one string
  getenv arg = one string
  file arg = filename
  atomfile arg = filename
  equal or atom args = one formula containing numbers, thermo keywords, math operations, g
    numbers = 0.0, 100, -5.4, 2.8e-4, etc
    constants = PI
    thermo keywords = vol, ke, etc from thermo style
    math operators = (), -x, x+y, x-y, x*y, x/y, x^y,
                     x==y, x!=y, xy, x>=y, x&&y, x||y, !x
    math functions = sqrt(x), exp(x), ln(x), log(x), abs(x),
                     sin(x), cos(x), tan(x), asin(x), acos(x), atan(x), atan2(y,x),
                     random(x,y,z), normal(x,y,z), ceil(x), floor(x), round(x)
                     ramp(x,y), stagger(x,y), logfreq(x,y,z), stride(x,y,z), vdisplace(x,y
    group functions = count(group), mass(group), charge(group),
                     xcm(group,dim), vcm(group,dim), fcm(group,dim),
                     bound(group,xmin), gyration(group), ke(group),
                     angmom(group,dim), torque(group,dim),
                     inertia(group,dimdim), omega(group,dim)
    region functions = count(group,region), mass(group,region), charge(group,region),
                     xcm(group,dim,region), vcm(group,dim,region), fcm(group,dim,region),
                     bound(group,xmin,region), gyration(group,region), ke(group,reigon),
                     angmom(group,dim,region), torque(group,dim,region),
                     inertia(group,dimdim,region), omega(group,dim,region)
    special functions = sum(x), min(x), max(x), ave(x), trap(x), gmask(x), rmask(x), grmas
    atom value = id[i], mass[i], type[i], x[i], y[i], z[i], vx[i], vy[i], vz[i], fx[i], fy
    atom vector = id, mass, type, x, y, z, vx, vy, vz, fx, fy, fz, omegax, omegay, omegaz,
    compute references = c_ID, c_ID[i], c_ID[i][j]
    fix references = f_ID, f_ID[i], f_ID[i][j]
    variable references = v_name, v_name[i]
```

Examples:

```
variable x index run1 run2 run3 run4 run5 run6 run7 run8
variable LoopVar loop $n
variable beta equal temp/3.0
variable b1 equal x[234]+0.5*vol
variable b1 equal "x[234] + 0.5*vol"
variable b equal xcm(mol1,x)/2.0
variable b equal c_myTemp
variable b atom x*y/vol
variable foo string myfile
variable f file values.txt
variable temp world 300.0 310.0 320.0 ${Tfinal}
variable x universe 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
variable x uloop 15 pad
variable x delete
```

Description:

This command assigns one or more strings to a variable name for evaluation later in the input script or during a simulation.

Variables can thus be useful in several contexts. A variable can be defined and then referenced elsewhere in an input script to become part of a new input command. For variable styles that store multiple strings, the next command can be used to increment which string is assigned to the variable. Variables of style *equal* store a formula which when evaluated produces a single numeric value which can be output either directly (see the print, fix print, and run every commands) or as part of thermodynamic output (see the thermo_style command), or used as input to an averaging fix (see the fix ave/time command). Variables of style *atom* store a formula which when evaluated produces one numeric value per atom which can be output to a dump file (see the dump custom command) or used as input to an averaging fix (see the fix ave/spatial and fix ave/atom commands). Variables of style *atomfile* can be used anywhere in an input script that atom-style variables are used; they get their per-atom values from a file rather than from a formula.

In the discussion that follows, the "name" of the variable is the arbitrary string that is the 1st argument in the variable command. This name can only contain alphanumeric characters and underscores. The "string" is one or more of the subsequent arguments. The "string" can be simple text as in the 1st example above, it can contain other variables as in the 2nd example, or it can be a formula as in the 3rd example. The "value" is the numeric quantity resulting from evaluation of the string. Note that the same string can generate different values when it is evaluated at different times during a simulation.

IMPORTANT NOTE: When the input script line is encountered that defines a variable of style *equal* or *atom* that contains a formula, the formula is NOT immediately evaluated and the result stored. See the discussion below about "Immediate Evaluation of Variables" if you want to do this.

IMPORTANT NOTE: When a variable command is encountered in the input script and the variable name has already been specified, the command is ignored. This means variables can NOT be re-defined in an input script (with 2 exceptions, read further). This is to allow an input script to be processed multiple times without resetting the variables; see the jump or include commands. It also means that using the command-line switch -var will override a corresponding index variable setting in the input script.

There are two exceptions to this rule. First, variables of style *string*, *getenv*, *equal* and *atom* ARE redefined each time the command is encountered. This allows these style of variables to be redefined multiple times in an input script. In a loop, this means the formula associated with an *equal* or *atom* style variable can change if it contains a substitution for another variable, e.g. $x.

Second, as described below, if a variable is iterated on to the end of its list of strings via the next command, it is removed from the list of active variables, and is thus available to be re-defined in a subsequent variable

variable command   615

command. The *delete* style does the same thing.

This section of the manual explains how occurrences of a variable name in an input script line are replaced by the variable's string. The variable name can be referenced as $x if the name "x" is a single character, or as ${LoopVar} if the name "LoopVar" is one or more characters.

As described below, for variable styles *index*, *loop*, *file*, *universe*, and *uloop*, which string is assigned to a variable can be incremented via the next command. When there are no more strings to assign, the variable is exhausted and a flag is set that causes the next jump command encountered in the input script to be skipped. This enables the construction of simple loops in the input script that are iterated over and then exited from.

As explained above, an exhausted variable can be re-used in an input script. The *delete* style also removes the variable, the same as if it were exhausted, allowing it to be redefined later in the input script or when the input script is looped over. This can be useful when breaking out of a loop via the if and jump commands before the variable would become exhausted. For example,

```
label      loop
variable   a loop 5
print      "A = $a"
if         "$a > 2" then "jump in.script break"
next       a
jump       in.script loop
label      break
variable   a delete
```

For the *index* style, one or more strings are specified. Initially, the 1st string is assigned to the variable. Each time a next command is used with the variable name, the next string is assigned. All processors assign the same string to the variable.

Index style variables with a single string value can also be set by using the command-line switch -var; see this section for details.

The *loop* style is identical to the *index* style except that the strings are the integers from 1 to N inclusive, if only one argument N is specified. This allows generation of a long list of runs (e.g. 1000) without having to list N strings in the input script. Initially, the string "1" is assigned to the variable. Each time a next command is used with the variable name, the next string ("2", "3", etc) is assigned. All processors assign the same string to the variable. The *loop* style can also be specified with two arguments N1 and N2. In this case the loop runs from N1 to N2 inclusive, and the string N1 is initially assigned to the variable. N1 <= N2 and N2 >= 0 is required.

For the *world* style, one or more strings are specified. There must be one string for each processor partition or "world". See this section of the manual for information on running LIGGGHTS(R)-WSU with multiple partitions via the "-partition" command-line switch. This variable command assigns one string to each world. All processors in the world are assigned the same string. The next command cannot be used with *equal* style variables, since there is only one value per world. This style of variable is useful when you wish to run different simulations on different partitions, or when performing a parallel tempering simulation (see the temper command), to assign different temperatures to different partitions.

For the *universe* style, one or more strings are specified. There must be at least as many strings as there are processor partitions or "worlds". See this page for information on running LIGGGHTS(R)-WSU with multiple partitions via the "-partition" command-line switch. This variable command initially assigns one string to each world. When a next command is encountered using this variable, the first processor partition to encounter it, is assigned the next available string. This continues until all the variable strings are consumed. Thus, this command can be used to run 50 simulations on 8 processor partitions. The simulations will be run one after the other on whatever partition becomes available, until they are all finished. *Universe* style variables are incremented using the files "tmp.lammps.variable" and "tmp.lammps.variable.lock" which you will see in

your directory during such a LIGGGHTS(R)-WSU run.

The *uloop* style is identical to the *universe* style except that the strings are the integers from 1 to N. This allows generation of long list of runs (e.g. 1000) without having to list N strings in the input script.

For the *string* style, a single string is assigned to the variable. The only difference between this and using the *index* style with a single string is that a variable with *string* style can be redefined. E.g. by another command later in the input script, or if the script is read again in a loop.

For the *getenv* style, a single string is assigned to the variable which should be the name of an environment variable. When the variable is evaluated, it returns the value of the environment variable, or an empty string if it not defined. This style of variable can be used to adapt the behavior of LIGGGHTS(R)-WSU input scripts via environment variable settings, or to retrieve information that has been previously stored with the shell putenv command. Note that because environment variable settings are stored by the operating systems, they persist beyond a clear command.

For the *file* style, a filename is provided which contains a list of strings to assign to the variable, one per line. The strings can be numeric values if desired. See the discussion of the next() function below for equal-style variables, which will convert the string of a file-style variable into a numeric value in a formula.

When a file-style variable is defined, the file is opened and the string on the first line is read and stored with the variable. This means the variable can then be evaluated as many times as desired and will return that string. There are two ways to cause the next string from the file to be read: use the next command or the next() function in an equal- or atom-style variable, as discussed below.

The rules for formatting the file are as follows. A comment character "#" can be used anywhere on a line; text starting with the comment character is stripped. Blank lines are skipped. The first "word" of a non-blank line, delimited by white space, is the "string" assigned to the variable.

For the *atomfile* style, a filename is provided which contains one or more sets of values, to assign on a per-atom basis to the variable. The format of the file is described below.

When an atomfile-style variable is defined, the file is opened and the first set of per-atom values are read and stored with the variable. This means the variable can then be evaluated as many times as desired and will return those values. There are two ways to cause the next set of per-atom values from the file to be read: use the next command or the next() function in an atom-style variable, as discussed below.

The rules for formatting the file are as follows. Each time a set of per-atom values is read, a non-blank line is searched for in the file. A comment character "#" can be used anywhere on a line; text starting with the comment character is stripped. Blank lines are skipped. The first "word" of a non-blank line, delimited by white space, is read as the count N of per-atom lines to immediately follow. N can be be the total number of atoms in the system, or only a subset. The next N lines have the following format

```
ID value
```

where ID is an atom ID and value is the per-atom numeric value that will be assigned to that atom. IDs can be listed in any order.

IMPORTANT NOTE: Every time a set of per-atom lines is read, the value for all atoms is first set to 0.0. Thus values for atoms whose ID does not appear in the set, will remain 0.0.

For the *equal* and *atom* styles, a single string is specified which represents a formula that will be evaluated afresh each time the variable is used. If you want spaces in the string, enclose it in double quotes so the parser will treat it as a single argument. For *equal* style variables the formula computes a scalar quantity, which becomes the value of the variable whenever it is evaluated. For *atom* style variables the formula computes one

variable command

617

quantity for each atom whenever it is evaluated.

Note that *equal* and *atom* variables can produce different values at different stages of the input script or at different times during a run. For example, if an *equal* variable is used in a [fix print](#) command, different values could be printed each timestep it was invoked. If you want a variable to be evaluated immediately, so that the result is stored by the variable instead of the string, see the section below on "Immediate Evaluation of Variables".

The next command cannot be used with *equal* or *atom* style variables, since there is only one string.

The formula for an *equal* or *atom* variable can contain a variety of quantities. The syntax for each kind of quantity is simple, but multiple quantities can be nested and combined in various ways to build up formulas of arbitrary complexity. For example, this is a valid (though strange) variable formula:

```
variable x equal "pe + c_MyTemp / vol^(1/3)"
```

Specifically, an formula can contain numbers, thermo keywords, math operators, math functions, group functions, region functions, atom values, atom vectors, compute references, fix references, and references to other variables.

Number	0.2, 100, 1.0e20, -15.4, etc
Constant	PI
Thermo keywords	vol, pe, ebond, etc
Math operators	(), -x, x+y, x-y, x*y, x/y, x^y, x==y, x!=y, xy, x>=y, x&&y, x||y, !x
Math functions	sqrt(x), exp(x), ln(x), log(x), abs(x), sin(x), cos(x), tan(x), asin(x), acos(x), atan(x), atan2(y,x), random(x,y,z), normal(x,y,z), ceil(x), floor(x), round(x), ramp(x,y), stagger(x,y), logfreq(x,y,z), stride(x,y,z), vdisplace(x,y), swiggle(x,y,z), cwiggle(x,y,z)
Group functions	count(ID), mass(ID), charge(ID), xcm(ID,dim), vcm(ID,dim), fcm(ID,dim), bound(ID,dir), gyration(ID), ke(ID), angmom(ID,dim), torque(ID,dim), inertia(ID,dimdim), omega(ID,dim)
Region functions	count(ID,IDR), mass(ID,IDR), charge(ID,IDR), xcm(ID,dim,IDR), vcm(ID,dim,IDR), fcm(ID,dim,IDR), bound(ID,dir,IDR), gyration(ID,IDR), ke(ID,IDR), angmom(ID,dim,IDR), torque(ID,dim,IDR), inertia(ID,dimdim,IDR), omega(ID,dim,IDR)
Special functions	sum(x), min(x), max(x), ave(x), trap(x), gmask(x), rmask(x), grmask(x,y), next(x)
Atom values	id[i], mass[i], type[i], x[i], y[i], z[i], vx[i], vy[i], vz[i], fx[i], fy[i], fz[i], omegax[i], omegay[i], omegaz[z], tqx[i], tqy[i], tqz[i], r[i], density[i]
Atom vectors	id, mass, type, x, y, z, vx, vy, vz, fx, fy, fz, omegax, omegay, omegaz, r, density
Compute references	c_ID, c_ID[i], c_ID[i][j]
Fix references	f_ID, f_ID[i], f_ID[i][j]
Other variables	v_name, v_name[i]

Most of the formula elements produce a scalar value. A few produce a per-atom vector of values. These are the atom vectors, compute references that represent a per-atom vector, fix references that represent a per-atom vector, and variables that are atom-style variables. Math functions that operate on scalar values produce a scalar value; math function that operate on per-atom vectors do so element-by-element and produce a per-atom vector.

A formula for equal-style variables cannot use any formula element that produces a per-atom vector. A formula for an atom-style variable can use formula elements that produce either a scalar value or a per-atom vector. Atom-style variables are evaluated by other commands that define a [group](#) on which they operate, e.g. a [dump](#) or [compute](#) or [fix](#) command. When they invoke the atom-style variable, only atoms in the group are inlcuded in the formula evaluation. The variable evaluates to 0.0 for atoms not in the group.

The thermo keywords allowed in a formula are those defined by the [thermo_style custom](#) command. Thermo keywords that require a [compute](#) to calculate their values such as "temp" or "press", use computes stored and invoked by the [thermo_style](#) command. This means that you can only use those keywords in a variable if the style you are using with the thermo_style command (and the thermo keywords associated with that style) also define and use the needed compute. Note that some thermo keywords use a compute indirectly to calculate their value (e.g. the enthalpy keyword uses temp, pe, and pressure). If a variable is evaluated directly in an input script (not during a run), then the values accessed by the thermo keyword must be current. See the discussion below about "Variable Accuracy".

Math Operators

Math operators are written in the usual way, where the "x" and "y" in the examples can themselves be arbitrarily complex formulas, as in the examples above. In this syntax, "x" and "y" can be scalar values or per-atom vectors. For example, "ke/natoms" is the division of two scalars, where "vy+vz" is the element-by-element sum of two per-atom vectors of y and z velocities.

Operators are evaluated left to right and have the usual C-style precedence: unary minus and unary logical NOT operator "!" have the highest precedence, exponentiation "^" is next; multiplication and division are next; addition and subtraction are next; the 4 relational operators "", and ">=" are next; the two remaining relational operators "==" and "!=" are next; then the logical AND operator "&&"; and finally the logical OR operator "||" has the lowest precedence. Parenthesis can be used to group one or more portions of a formula and/or enforce a different order of evaluation than what would occur with the default precedence.

IMPORTANT NOTE: Because a unary minus is higher precedence than exponentiation, the formula "-2^2" will evaluate to 4, not -4. This convention is compatible with some programming languages, but not others. As mentioned, this behavior can be easily overridden with parenthesis; the formula "-(2^2)" will evaluate to -4.

The 6 relational operators return either a 1.0 or 0.0 depending on whether the relationship between x and y is TRUE or FALSE. For example the expression x

These relational and logical operators can be used as a masking or selection operation in a formula. For example, the number of atoms whose properties satifsy one or more criteria could be calculated by taking the returned per-atom vector of ones and zeroes and passing it to the [compute reduce](#) command.

Math Functions

Math functions are specified as keywords followed by one or more parenthesized arguments "x", "y", "z", each of which can themselves be arbitrarily complex formulas. In this syntax, the arguments can represent scalar values or per-atom vectors. In the latter case, the math operation is performed on each element of the vector. For example, "sqrt(natoms)" is the sqrt() of a scalar, where "sqrt(y*z)" yields a per-atom vector with each element being the sqrt() of the product of one atom's y and z coordinates.

Most of the math functions perform obvious operations. The ln() is the natural log; log() is the base 10 log.

The random(x,y,z) function takes 3 arguments: x = lo, y = hi, and z = seed. It generates a uniform random number between lo and hi. The normal(x,y,z) function also takes 3 arguments: x = mu, y = sigma, and z = seed. It generates a Gaussian variate centered on mu with variance sigma^2. In both cases the seed is used the

first time the internal random number generator is invoked, to initialize it. For equal-style variables, every processor uses the same seed so that they each generate the same sequence of random numbers. For atom-style variables, a unique seed is created for each processor, based on the specified seed. This effectively generates a different random number for each atom being looped over in the atom-style variable.

IMPORTANT NOTE: Internally, there is just one random number generator for all equal-style variables and one for all atom-style variables. If you define multiple variables (of each style) which use the random() or normal() math functions, then the internal random number generators will only be initialized once, which means only one of the specified seeds will determine the sequence of generated random numbers.

The ceil(), floor(), and round() functions are those in the C math library. Ceil() is the smallest integer not less than its argument. Floor() if the largest integer not greater than its argument. Round() is the nearest integer to its argument.

The ramp(x,y) function uses the current timestep to generate a value linearly intepolated between the specified x,y values over the course of a run, according to this formula:

```
value = x + (y-x) * (timestep-startstep) / (stopstep-startstep)
```

The run begins on startstep and ends on stopstep. Startstep and stopstep can span multiple runs, using the *start* and *stop* keywords of the run command. See the run command for details of how to do this.

The stagger(x,y) function uses the current timestep to generate a new timestep. X,y > 0 and x > y are required. The generated timesteps increase in a staggered fashion, as the sequence x,x+y,2x,2x+y,3x,3x+y,etc. For any current timestep, the next timestep in the sequence is returned. Thus if stagger(1000,100) is used in a variable by the dump_modify every command, it will generate the sequence of output timesteps:

```
100,1000,1100,2000,2100,3000,etc
```

The logfreq(x,y,z) function uses the current timestep to generate a new timestep. X,y,z > 0 and y < z are required. The generated timesteps increase in a logarithmic fashion, as the sequence x,2x,3x,...y*x,z*x,2*z*x,3*z*x,...y*z*x,z*z*x,2*z*x*x,etc. For any current timestep, the next timestep in the sequence is returned. Thus if logfreq(100,4,10) is used in a variable by the dump_modify every command, it will generate the sequence of output timesteps:

```
100,200,300,400,1000,2000,3000,4000,10000,20000,etc
```

The stride(x,y,z) function uses the current timestep to generate a new timestep. X,y >= 0 and z > 0 and x <= y are required. The generated timesteps increase in increments of z, from x to y, I.e. it generates the sequece x,x+z,x+2z,...,y. If y-x is not a multiple of z, then similar to the way a for loop operates, the last value will be one that does not exceed y. For any current timestep, the next timestep in the sequence is returned. Thus if stagger(1000,2000,100) is used in a variable by the dump_modify every command, it will generate the sequence of output timesteps:

```
1000,1100,1200, ... ,1900,2000
```

The vdisplace(x,y) function takes 2 arguments: x = value0 and y = velocity, and uses the elapsed time to change the value by a linear displacement due to the applied velocity over the course of a run, according to this formula:

```
value = value0 + velocity*(timestep-startstep)*dt
```

where dt = the timestep size.

The run begins on startstep. Startstep can span multiple runs, using the *start* keyword of the run command.

Math Functions   620

See the run command for details of how to do this. Note that the thermo_style keyword elaplong = timestep-startstep.

The swiggle(x,y,z) and cwiggle(x,y,z) functions each take 3 arguments: x = value0, y = amplitude, z = period. They use the elapsed time to oscillate the value by a sin() or cos() function over the course of a run, according to one of these formulas, where omega = 2 PI / period:

```
value = value0 + Amplitude * sin(omega*(timestep-startstep)*dt)
value = value0 + Amplitude * (1 - cos(omega*(timestep-startstep)*dt))
```

where dt = the timestep size.

The run begins on startstep. Startstep can span multiple runs, using the *start* keyword of the run command. See the run command for details of how to do this. Note that the thermo_style keyword elaplong = timestep-startstep.

Group and Region Functions

Group functions are specified as keywords followed by one or two parenthesized arguments. The first argument is the group-ID. The *dim* argument, if it exists, is *x* or *y* or *z*. The *dir* argument, if it exists, is *xmin*, *xmax*, *ymin*, *ymax*, *zmin*, or *zmax*. The *dimdim* argument, if it exists, is *xx* or *yy* or *zz* or *xy* or *yz* or *xz*.

The group function count() is the number of atoms in the group. The group functions mass() and charge() are the total mass and charge of the group. Xcm() and vcm() return components of the position and velocity of the center of mass of the group. Fcm() returns a component of the total force on the group of atoms. Bound() returns the min/max of a particular coordinate for all atoms in the group. Gyration() computes the radius-of-gyration of the group of atoms. See the compute gyration command for a definition of the formula. Angmom() returns components of the angular momentum of the group of atoms around its center of mass. Torque() returns components of the torque on the group of atoms around its center of mass, based on current forces on the atoms. Inertia() returns one of 6 components of the symmetric inertia tensor of the group of atoms around its center of mass, ordered as Ixx,Iyy,Izz,Ixy,Iyz,Ixz. Omega() returns components of the angular velocity of the group of atoms around its center of mass.

Region functions are specified exactly the same way as group functions except they take an extra argument which is the region ID. The function is computed for all atoms that are in both the group and the region. If the group is "all", then the only criteria for atom inclusion is that it be in the region.

Special Functions

Special functions take specific kinds of arguments, meaning their arguments cannot be formulas themselves.

The sum(x), min(x), max(x), ave(x), and trap(x) functions each take 1 argument which is of the form "c_ID" or "c_ID[N]" or "f_ID" or "f_ID[N]". The first two are computes and the second two are fixes; the ID in the reference should be replaced by the ID of a compute or fix defined elsewhere in the input script. The compute or fix must produce either a global vector or array. If it produces a global vector, then the notation without "[N]" should be used. If it produces a global array, then the notation with "[N]" should be used, when N is an integer, to specify which column of the global array is being referenced.

These functions operate on the global vector of inputs and reduce it to a single scalar value. This is analagous to the operation of the compute reduce command, which invokes the same functions on per-atom and local vectors.

The sum() function calculates the sum of all the vector elements. The min() and max() functions find the minimum and maximum element respectively. The ave() function is the same as sum() except that it divides the result by the length of the vector. The trap() function is the same as sum() except the first and last elements

are multiplied by a weighting factor of 1/2 when performing the sum. This effectively implements an integratiion via the trapezoidal rule on the global vector of data. I.e. consider a set of points, equally spaced by 1 in their x coordinate: (1,V1), (2,V2), ..., (N,VN), where the Vi are the values in the global vector of length N. The integral from 1 to N of these points is trap(). When appropriately normalized by the timestep size, this function is useful for calculating integrals of time-series data, like that generated by the fix ave/correlate command.

The gmask(x) function takes 1 argument which is a group ID. It can only be used in atom-style variables. It returns a 1 for atoms that are in the group, and a 0 for atoms that are not.

The rmask(x) function takes 1 argument which is a region ID. It can only be used in atom-style variables. It returns a 1 for atoms that are in the geometric region, and a 0 for atoms that are not.

The grmask(x,y) function takes 2 arguments. The first is a group ID, and the second is a region ID. It can only be used in atom-style variables. It returns a 1 for atoms that are in both the group and region, and a 0 for atoms that are not in both.

The next(x) function takes 1 argument which is a variable ID (not "v_foo", just "foo"). It must be for a file-style or atomfile-style variable. Each time the next() function is invoked (i.e. each time the equal-style or atom-style variable is evaluated), the following steps occur.

For file-style variables, the current string value stored by the file-style variable is converted to a numeric value and returned by the function. And the next string value in the file is read and stored. Note that if the line previously read from the file was not a numeric string, then it will typically evaluate to 0.0, which is likely not what you want.

For atomfile-style variables, the current per-atom values stored by the atomfile-style variable are returned by the function. And the next set of per-atom values in the file is read and stored.

Since file-style and atomfile-style variables read and store the first line of the file or first set of per-atoms values when they are defined in the input script, these are the value(s) that will be returned the first time the next() function is invoked. If next() is invoked more times than there are lines or sets of lines in the file, the variable is deleted, similar to how the next command operates.

Atom Values and Vectors

Atom values take a single integer argument I from 1 to N, where I is the an atom-ID, e.g. x[243], which means use the x coordinate of the atom with ID = 243.

Atom vectors generate one value per atom, so that a reference like "vx" means the x-component of each atom's velocity will be used when evaluating the variable. Note that other atom attributes can be used as inputs to a variable by using the compute property/atom command and then specifying a quantity from that compute.

Compute References

Compute references access quantities calculated by a compute. The ID in the reference should be replaced by the ID of a compute defined elsewhere in the input script. As discussed in the doc page for the compute command, computes can produce global, per-atom, or local values. Only global and per-atom values can be used in a variable. Computes can also produce a scalar, vector, or array. An equal-style variable can only use scalar values, which means a global scalar, or an element of a global or per-atom vector or array. Atom-style variables can use the same scalar values. They can also use per-atom vector values. A vector value can be a per-atom vector itself, or a column of an per-atom array. See the doc pages for individual computes to see what kind of values they produce.

Examples of different kinds of compute references are as follows. There is no ambiguity as to what a reference means, since computes only produce global or per-atom quantities, never both.

c_ID	global scalar, or per-atom vector
c_ID[I]	Ith element of global vector, or atom I's value in per-atom vector, or Ith column from per-atom array
c_ID[I][J]	I,J element of global array, or atom I's Jth value in per-atom array

If a variable containing a compute is evaluated directly in an input script (not during a run), then the values accessed by the compute must be current. See the discussion below about "Variable Accuracy".

Fix References

Fix references access quantities calculated by a fix. The ID in the reference should be replaced by the ID of a fix defined elsewhere in the input script. As discussed in the doc page for the fix command, fixes can produce global, per-atom, or local values. Only global and per-atom values can be used in a variable. Fixes can also produce a scalar, vector, or array. An equal-style variable can only use scalar values, which means a global scalar, or an element of a global or per-atom vector or array. Atom-style variables can use the same scalar values. They can also use per-atom vector values. A vector value can be a per-atom vector itself, or a column of an per-atom array. See the doc pages for individual fixes to see what kind of values they produce.

The different kinds of fix references are exactly the same as the compute references listed in the above table, where "c_" is replaced by "f_". Again, there is no ambiguity as to what a reference means, since fixes only produce global or per-atom quantities, never both.

f_ID	global scalar, or per-atom vector
f_ID[I]	Ith element of global vector, or atom I's value in per-atom vector, or Ith column from per-atom array
f_ID[I][J]	I,J element of global array, or atom I's Jth value in per-atom array

If a variable containing a fix is evaluated directly in an input script (not during a run), then the values accessed by the fix should be current. See the discussion below about "Variable Accuracy".

Note that some fixes only generate quantities on certain timesteps. If a variable attempts to access the fix on non-allowed timesteps, an error is generated. For example, the fix ave/time command may only generate averaged quantities every 100 steps. See the doc pages for individual fix commands for details.

Variable References

Variable references access quantities stored or calculated by other variables, which will cause those variables to be evaluated. The name in the reference should be replaced by the name of a variable defined elsewhere in the input script.

As discussed on this doc page, equal-style variables generate a global scalar numeric value; atom-style and atomfile-style variables generate a per-atom vector of numeric values; all other variables store a string. The formula for an equal-style variable can use any style of variable except an atom-style or atomfile-style (unless only a single value from the variable is accessed via a subscript). If a string-storing variable is used, the string is converted to a numeric value. Note that this will typically produce a 0.0 if the string is not a numeric string, which is likely not what you want. The formula for an atom-style variable can use any style of variable, including other atom-style or atomfile-style variables.

Examples of different kinds of variable references are as follows. There is no ambiguity as to what a reference means, since variables produce only a global scalar or a per-atom vector, never both.

v_name	scalar, or per-atom vector

v_name[I]	atom I's value in per-atom vector

Immediate Evaluation of Variables:

There is a difference between referencing a variable with a leading $ sign (e.g. $x or ${abc}) versus with a leading "v_" (e.g. v_x or v_abc). The former can be used in any input script command, including a variable command. The input script parser evaluates the reference variable immediately and substitutes its value into the command. As explained in [Section commands 3.2](#) for "Parsing rules", you can also use un-named "immediate" variables for this purpose. An variable reference such as $((xlo+xhi)/2+sqrt(v_area)) evaluates the string between the parenthesis as an equal-style variable.

Referencing a variable with a leading "v_" is an optional or required kind of argument for some commands (e.g. the [fix ave/spatial](#) or [dump custom](#) or [thermo_style](#) commands) if you wish it to evaluate a variable periodically during a run. It can also be used in a variable formula if you wish to reference a second variable. The second variable will be evaluated whenever the first variable is evaluated.

As an example, suppose you use this command in your input script to define the variable "v" as

```
variable v equal vol
```

before a run where the simulation box size changes. You might think this will assign the initial volume to the variable "v". That is not the case. Rather it assigns a formula which evaluates the volume (using the thermo_style keyword "vol") to the variable "v". If you use the variable "v" in some other command like [fix ave/time](#) then the current volume of the box will be evaluated continuously during the run.

If you want to store the initial volume of the system, you can do it this way:

```
variable v equal vol
variable v0 equal $v
```

The second command will force "v" to be evaluated (yielding the initial volume) and assign that value to the variable "v0". Thus the command

```
thermo_style custom step v_v v_v0
```

would print out both the current and initial volume periodically during the run.

Note that it is a mistake to enclose a variable formula in double quotes if it contains variables preceeded by $ signs. For example,

```
variable vratio equal "${vfinal}/${v0}"
```

This is because the quotes prevent variable substitution (see [this section](#) on parsing input script commands), and thus an error will occur when the formula for "vratio" is evaluated later.

Variable Accuracy:

Obviously, LIGGGHTS(R)-WSU attempts to evaluate variables containing formulas (*equal* and *atom* style variables) accurately whenever the evaluation is performed. Depending on what is included in the formula, this may require invoking a [compute](#), either directly or indirectly via a thermo keyword, or accessing a value previously calculated by a compute, or accessing a value calculated and stored by a [fix](#). If the compute is one that calculates the pressure or energy of the system, then these quantities need to be tallied during the evaluation of the interatomic potentials (pair, bond, etc) on timesteps that the variable will need the values.

LIGGGHTS(R)-WSU keeps track of all of this during a run or energy minimization. An error will be generated if you attempt to evaluate a variable on timesteps when it cannot produce accurate values. For example, if a thermo_style custom command prints a variable which accesses values stored by a fix ave/time command and the timesteps on which thermo output is generated are not multiples of the averaging frequency used in the fix command, then an error will occur.

An input script can also request variables be evaluated before or after or in between runs, e.g. by including them in a print command. In this case, if a compute is needed to evaluate a variable (either directly or indirectly), LIGGGHTS(R)-WSU will not invoke the compute, but it will use a value previously calculated by the compute, and can do this only if it is current. Fixes will always provide a quantity needed by a variable, but the quantity may or may not be current. This leads to one of three kinds of behavior:

(1) The variable may be evaluated accurately. If it contains references to a compute or fix, and these values were calculated on the last timestep of a preceeding run, then they will be accessed and used by the variable and the result will be accurate.

(2) LIGGGHTS(R)-WSU may not be able to evaluate the variable and will generate an error message stating so. For example, if the variable requires a quantity from a compute that is not current, LIGGGHTS(R)-WSU will generate an error. This means, for example, that such a variable cannot be evaluated before the first run has occurred. Likewise, in between runs, such a variable cannot be accessed unless it was evaluated on the last timestep of the preceding run, e.g. by thermodynamic output.

One way to get around this problem is to perform a 0-timestep run before using the variable. For example, these commands

```
variable t equal temp
print "Initial temperature = $t"
run 1000
```

will generate an error if the run is the first run specified in the input script, because generating a value for the "t" variable requires a compute for calculating the temperature to be invoked.

However, this sequence of commands would be fine:

```
run 0
variable t equal temp
print "Initial temperature = $t"
run 1000
```

The 0-timestep run initializes and invokes various computes, including the one for temperature, so that the value it stores is current and can be accessed by the variable "t" after the run has completed. Note that a 0-timestep run does not alter the state of the system, so it does not change the input state for the 1000-timestep run that follows. Also note that the 0-timestep run must actually use and invoke the compute in question (e.g. via thermo or dump output) in order for it to enable the compute to be used in a variable after the run. Thus if you are trying to print a variable that uses a compute you have defined, you could insure it was invoked on the last timestep of the preceding run by including it in thermodynamic output.

Unlike computes, fixes will never generate an error if their values are accessed by a variable in between runs. They always return some value to the variable. However, the value may not be what you expect if the fix has not yet calculated the quantity of interest or it is not current. For example, the fix indent command stores the force on the indenter. But this is not computed until a run is performed. Thus if a variable attempts to print this value before the first run, zeroes will be output. Again, performing a 0-timestep run before printing the variable has the desired effect.

(3) The variable may be evaluated incorrectly and LIGGGHTS(R)-WSU may have no way to detect this has occurred. Consider the following sequence of commands:

```
pair_coeff 1 1 1.0 1.0
run 1000
pair_coeff 1 1 1.5 1.0
variable e equal pe
print "Final potential energy = $e"
```

The first run is performed using one setting for the pairwise potential defined by the pair_style and pair_coeff commands. The potential energy is evaluated on the final timestep and stored by the compute pe compute (this is done by the thermo_style command). Then a pair coefficient is changed, altering the potential energy of the system. When the potential energy is printed via the "e" variable, LIGGGHTS(R)-WSU will use the potential energy value stored by the compute pe compute, thinking it is current. There are many other commands which could alter the state of the system between runs, causing a variable to evaluate incorrectly.

The solution to this issue is the same as for case (2) above, namely perform a 0-timestep run before the variable is evaluated to insure the system is up-to-date. For example, this sequence of commands would print a potential energy that reflected the changed pairwise coefficient:

```
pair_coeff 1 1 1.0 1.0
run 1000
pair_coeff 1 1 1.5 1.0
run 0
variable e equal pe
print "Final potential energy = $e"
```

Restrictions:

Indexing any formula element by global atom ID, such as an atom value, requires the atom style to use a global mapping in order to look up the vector indices. By default, only atom styles with molecular information create global maps. The atom_modify map command can override the default.

All *universe-* and *uloop*-style variables defined in an input script must have the same number of values.

Related commands:

next, jump, include, temper, fix print, print

Default: none

velocity command

Syntax:

```
velocity group-ID style args keyword value ...
```

- group-ID = ID of group of atoms whose velocity will be changed
- style = *set* or *ramp* or *zero*

```
  set args = vx vy vz
      vx,vy,vz = velocity value or NULL (velocity units)
      any of vx,vy,vz van be a variable (see below)
    setAngular args = angularmx angularmy angularmz
      angularmx,angularmy,angularmz = angular momentum value or NULL (angular mometum units)
      any of angularmx,angularmy,angularmz van be a variable (see below)
    ramp args = vdim vlo vhi dim clo chi
      vdim = vx or vy or vz
      vlo,vhi = lower and upper velocity value (velocity units)
      dim = x or y or z
      clo,chi = lower and upper coordinate bound (distance units)
    zero arg = linear or angular
      linear = zero the linear momentum
      angular = zero the angular momentum
```

- zero or more keyword/value pairs may be appended
- keyword = *dist* or *sum* or *mom* or *rot* or *loop* or *units*

```
  dist value = uniform or gaussian
    sum value = no or yes
    mom value = no or yes
    rot value = no or yes
    loop value = all or local or geom
    rigid value = fix-ID
      fix-ID = ID of rigid body fix
    units value = box or lattice
```

Examples:

```
velocity border set NULL 4.0 v_vz sum yes units box
velocity all zero linear
```

Description:

Set or change the velocities of a group of atoms in one of several styles. For each style, there are required arguments and optional keyword/value parameters. Not all options are used by each style. Each option has a default as listed below.

The *set* style sets the velocities of all atoms in the group to the specified values. If any component is specified as NULL, then it is not set. Any of the vx,vy,vz velocity components can be specified as an equal-style or atom-style [variable](#). If the value is a variable, it should be specified as v_name, where name is the variable name. In this case, the variable will be evaluated, and its value used to determine the velocity component.

The *setAngular* is simular to *set*, however, sets the angular momentum.

Equal-style variables can specify formulas with various mathematical functions, and include [thermo_style](#) command keywords for the simulation box parameters or other parameters.

Atom-style variables can specify the same formulas as equal-style variables but can also include per-atom values, such as atom coordinates. Thus it is easy to specify a spatially-dependent velocity field.

The *ramp* style makekes velocities ramped uniformly from vlo to vhi are applied to dimension vx, or vy, or vz. The value assigned to a particular atom depends on its relative coordinate value (in dim) from clo to chi. For the example above, an atom with y-coordinate of 10 (1/4 of the way from 5 to 25), would be assigned a x-velocity of 1.25 (1/4 of the way from 0.0 to 5.0). Atoms outside the coordinate bounds (less than 5 or greater than 25 in this case), are assigned velocities equal to vlo or vhi (0.0 or 5.0 in this case).

The *zero* style adjusts the velocities of the group of atoms so that the aggregate linear or angular momentum is zero. No other changes are made to the velocities of the atoms. If the *rigid* option is specified (see below), then the zeroing is performed on individual rigid bodies, as defined by the [fix rigid or fix rigid/small](#) commands. In other words, zero linear will set the linear momentum of each rigid body to zero, and zero angular will set the angular momentum of each rigid body to zero. This is done by adjusting the velocities of the atoms in each rigid body.

For all styles, no atoms are assigned z-component velocities if the simulation is 2d; see the [dimension](#) command.

The keyword/value option pairs are used in the following ways by the various styles.

The *sum* option is used by all styles, except *zero*. The new velocities will be added to the existing ones if sum = yes, or will replace them if sum = no.

The *mom* and *rot* options are used by *create*. If mom = yes, the linear momentum of the newly created ensemble of velocities is zeroed; if rot = yes, the angular momentum is zeroed.

The *loop* option is used by *create* in the following ways.

If loop = all, then each processor loops over all atoms in the simulation to create velocities, but only stores velocities for atoms it owns. This can be a slow loop for a large simulation. If atoms were read from a data file, the velocity assigned to a particular atom will be the same, independent of how many processors are being used. This will not be the case if atoms were created using the [create_atoms](#) command, since atom IDs will likely be assigned to atoms differently.

If loop = local, then each processor loops over only its atoms to produce velocities. The random number seed is adjusted to give a different set of velocities on each processor. This is a fast loop, but the velocity assigned to a particular atom will depend on which processor owns it. Thus the results will always be different when a simulation is run on a different number of processors.

If loop = geom, then each processor loops over only its atoms. For each atom a unique random number seed is created, based on the atom's xyz coordinates. A velocity is generated using that seed. This is a fast loop and the velocity assigned to a particular atom will be the same, independent of how many processors are used. However, the set of generated velocities may be more correlated than if the *all* or *local* options are used.

Note that the *loop geom* option will not necessarily assign identical velocities for two simulations run on different machines. This is because the computations based on xyz coordinates are sensitive to tiny differences in the double-precision value for a coordinate as stored on a particular machine.

The *rigid* option only has meaning when used with the *zero* style. It allows specification of a fix-ID for one of the [rigid-body fix](#) variants which defines a set of rigid bodies. The zeroing of linear or angular momentum is then performed for each rigid body defined by the fix, as described above.

The *units* option is used by *set*, *setAngular* and *ramp*. If units = box, the velocities and coordinates specified in the velocity command are in the standard units described by the [units](#) command (e.g. Angstroms/fmsec for real units). If units = lattice, velocities are in units of lattice spacings per time (e.g. spacings/fmsec) and coordinates are in lattice spacings. The [lattice](#) command must have been previously used to define the lattice spacing.

Restrictions: none

Related commands:

[fix shake](#), [lattice](#)

Default:

The option defaults are dist = uniform, sum = no, mom = yes, rot = no, style on group-ID, loop = all, and units = box. The rigid option is not defined by default.

---

write_data command

Syntax:

```
write_data file keyword value ...
```

- file = name of data file to write out
- zero or more keyword/value pairs may be appended
- keyword = *pair*

```
 pair value = ii or ij
     ii = write one line of pair coefficient info per atom type
     ij = write one line of pair coefficient info per IJ atom type pair
```

Examples:

```
write_data data.*
```

Description:

Write a data file in text format of the current state of the simulation. Data files can be read by the [read data](#) command to begin a simulation. The [read_data](#) command also describes their format.

Similar to [dump](#) files, the data filename can contain a "*" wild-card character. The "*" is replaced with the current timestep value.

IMPORTANT NOTE: The write-data command is not yet fully implemented in two respects. First, most pair styles do not yet write their coefficient information into the data file. This means you will need to specify that information in your input script that reads the data file, via the [pair_coeff](#) command. Second, a few of the [atom styles](#) (ellipsoid, line, tri) that store auxiliary "bonus" information about aspherical particles, do not yet write the bonus info into the data file. Both these functionalities will be added to the write_data command later.

Because a data file is in text format, if you use a data file written out by this command to restart a simulation, the initial state of the new run will be slightly different than the final state of the old run (when the file was written) which was represented internally by LIGGGHTS(R)-WSU in binary format. A new simulation which reads the data file will thus typically diverge from a simulation that continued in the original input script.

If you want to do more exact restarts, using binary files, see the [restart](#), [write_restart](#), and [read_restart](#) commands. You can also convert binary restart files to text data files, after a simulation has run, using the [-r command-line switch](#).

IMPORTANT NOTE: Only limited information about a simulation is stored in a data file. For example, no information about atom [groups](#) and [fixes](#) are stored. [Binary restart files](#) store more information.

Bonds that are broken (e.g. by a bond-breaking potential) are not written to the data file. Thus these bonds will not exist when the data file is read.

The *pair* keyword lets you specify in what format the pair coefficient information is written into the data file. If the value is specified as *ii*, then one line per atom type is written, to specify the coefficients for each of the I=J interactions. This means that no cross-interactions for I != J will be specified in the data file and the pair style will apply its mixing rule, as documented on individual [pair_style](#) doc pages. Of course this behavior can

be overridden in the input script after reading the data file, by specifying additional pair_coeff commands for any desired I,J pairs.

If the value is specified as *ij*, then one line of coefficients is written for all I,J pairs where I <= J. These coefficients will include any specific settings made in the input script up to that point. The presence of these I != J coefficients in the data file will effectively turn off the default mixing rule for the pair style. Again, the coefficient values in the data file can can be overridden in the input script after reading the data file, by specifying additional pair_coeff commands for any desired I,J pairs.

Restrictions:

This command requires inter-processor communication to migrate atoms before the data file is written. This means that your system must be ready to perform a simulation before using this command (force fields setup, atom masses initialized, etc).

Related commands:

read_data, write_restart

Default:

The option defaults are pair = ii.

write_dump command

Syntax:

```
write_dump group-ID style file dump-args modify dump_modify-args
```

- group-ID = ID of the group of atoms to be dumped
- style = any of the supported [dump styles](#)
- file = name of file to write dump info to
- dump-args = any additional args needed for a particular [dump style](#)
- modify = all args after this keyword are passed to [dump_modify](#) (optional)
- dump-modify-args = args for [dump_modify](#) (optional)

Examples:

```
write_dump all atom dump.atom
write_dump subgroup atom dump.run.bin
write_dump all custom dump.myforce.* id type x y vx fx
write_dump flow custom dump.%.myforce id type c_myF[3] v_ke modify sort id
write_dump all xyz system.xyz modify sort id elements O H
write_dump all image snap*.jpg type type size 960 960 modify backcolor white
write_dump all image snap*.jpg element element &
   bond atom 0.3 shiny 0.1 ssao yes 6345 0.2 size 1600 1600  &
   modify backcolor white element C C O H N C C C O H H S O H
```

Description:

Dump a single snapshot of atom quantities to one or more files for the current state of the system. This is a one-time immediate operation, in contrast to the [dump](#) command which will will set up a dump style to write out snapshots periodically during a running simulation.

The syntax for this command is mostly identical to that of the [dump](#) and [dump_modify](#) commands as if they were concatenated together, with the following exceptions: There is no need for a dump ID or dump frequency and the keyword *modify* is added. The latter is so that the full range of [dump_modify](#) options can be specified for the single snapshot, just as they can be for multiple snapshots. The *modify* keyword separates the arguments that would normally be passed to the *dump* command from those that would be given the *dump_modify*. Both support optional arguments and thus LIGGGHTS(R)-WSU needs to be able to cleanly separate the two sets of args.

Note that if the specified filename uses wildcard characters "*" or "%", as supported by the [dump](#) commmand, they will operate in the same fashion to create the new filename(s). Normally, [dump image](#) files require a filename with a "*" character for the timestep. That is not the case for the write_dump command; no wildcard "*" character is necessary.

Restrictions:

All restrictions for the [dump](#) and [dump_modify](#) commands apply to this command as well, with the exception of the [dump image](#) filename not requiring a wildcard "*" character, as noted above.

Since dumps are normally written during a [run](#) or [energy minimization](#), the simulation has to be ready to run before this command can be used. Similarly, if the dump requires information from a compute, fix, or variable, the information needs to have been calculated for the current timestep (e.g. by a prior run), else LIGGGHTS(R)-WSU will generate an error message.

For example, it is not possible to dump per-atom energy with this command before a run has been performed, since no energies and forces have yet been calculated. See the [variable](#) doc page sectinn on Variable Accuracy for more information on this topic.

Related commands:

[dump](#), [dump image](#), [dump_modify](#)

Default:

The defaults are listed on the doc pages for the [dump](#) and [dump image](#) and [dump_modify](#) commands.

write_restart command

Syntax:

```
write_restart file keyword value ...
```

- file = name of file to write restart information to
- zero or more keyword/value pairs may be appended
- keyword = *fileper* or *nfile*

```
   fileper arg = Np
      Np = write one file for every this many processors
    nfile arg = Nf
      Nf = write this many files, one from each of Nf processors
```

Examples:

```
write_restart restart.equil
write_restart poly.%.* nfile 10
```

Description:

Write a binary restart file of the current state of the simulation.

During a long simulation, the restart command is typically used to output restart files periodically. The write_restart command is useful after a minimization or whenever you wish to write out a single current restart file.

Similar to dump files, the restart filename can contain two wild-card characters. If a "*" appears in the filename, it is replaced with the current timestep value. If a "%" character appears in the filename, then one file is written by each processor and the "%" character is replaced with the processor ID from 0 to P-1. An additional file with the "%" replaced by "base" is also written, which contains global information. For example, the files written for filename restart.% would be restart.base, restart.0, restart.1, ... restart.P-1. This creates smaller files and can be a fast mode of output and subsequent input on parallel machines that support parallel I/O. The optional *fileper* and *nfile* keywords discussed below can alter the number of files written.

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, you can use the -r command-line switch to convert a restart file to a data file.

IMPORTANT NOTE: Although the purpose of restart files is to enable restarting a simulation from where it left off, not all information about a simulation is stored in the file. For example, the list of fixes that were specified during the initial run is not stored, which means the new input script must specify any fixes you want to use. Even when restart information is stored in the file, as it is for some fixes, commands may need to be re-specified in the new input script, in order to re-use that information. See the read_restart command for information about what is stored in a restart file.

The optional *nfile* or *fileper* keywords can be used in conjunction with the "%" wildcard character in the specified restart file name. As explained above, the "%" character causes the restart file to be written in pieces, one piece for each of P processors. By default P = the number of processors the simulation is running on. The *nfile* or *fileper* keyword can be used to set P to a smaller value, which can be more efficient when running on a large number of processors.

The *nfile* keyword sets P to the specified Nf value. For example, if Nf = 4, and the simulation is running on 100 processors, 4 files will be written, by processors 0,25,50,75. Each will collect information from itself and the next 24 processors and write it to a restart file.

For the *fileper* keyword, the specified value of Np means write one file for every Np processors. For example, if Np = 4, every 4th processor (0,4,8,12,etc) will collect information from itself and the next 3 processors and write it to a restart file.

Restrictions:

This command requires inter-processor communication to migrate atoms before the restart file is written. This means that your system must be ready to perform a simulation before using this command (force fields setup, atom masses initialized, etc).

Related commands:

restart, read_restart, write_data

Default: none