

2.1 fix accelerate/cos command

2.1.1 Syntax

```
fix ID group-ID accelerate value
```

- ID, group-ID are documented in [fix](#) command
- accelerate/cos = style name of this fix command
- value = amplitude of acceleration (in unit of velocity/time)

2.1.2 Examples

```
fix 1 all accelerate/cos 2.0e-7
```

2.1.3 Description

Give each atom a acceleration in x-direction based on its z coordinate. The acceleration is a periodic function along the z-direction:

$$a_x(z) = A \cos\left(\frac{2\pi z}{l_z}\right)$$

where A is the acceleration amplitude, l_z is the z-length of the simulation box. At steady state, the acceleration generates a velocity profile:

$$v_x(z) = V \cos\left(\frac{2\pi z}{l_z}\right)$$

The generated velocity amplitude V is related to the shear viscosity η by:

$$V = \frac{A\rho}{\eta} \left(\frac{l_z}{2\pi}\right)^2$$

and it can be obtained from ensemble average of the velocity profile:

$$V = \frac{\sum_i 2m_i v_{i,x} \cos\left(\frac{2\pi z_i}{l_z}\right)}{\sum_i m_i},$$

where m_i , $v_{i,x}$, and z_i are the mass, x -component velocity, and z -coordinate of a particle, respectively.

The velocity amplitude V can be calculated with [*compute viscosity/cos*](#), which enables viscosity calculation with periodic perturbation method, as described by [*Hess*](#). Because the applied acceleration drives the system away from equilibration, the calculated shear viscosity is lower than the intrinsic viscosity due to the shear-thinning effect. Extrapolation to zero acceleration should generally be performed to predict the zero-shear viscosity. As the shear stress decreases, the signal-to-noise ratio decreases rapidly, and the simulation time must be extended accordingly to get converged results.

In order to get meaningful results, the group ID of this fix should be all.

2.1.4 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.

2.1.5 Restrictions

This fix is part of the MISC package. It is only enabled if LAMMPS was built with that package. See the [*Build package*](#) page for more info.

Since this fix depends on the z -coordinate of atoms, it cannot be used in 2d simulations.

2.1.6 Related commands

[*compute viscosity/cos*](#)

2.1.7 Default

none

(**Hess**) Hess, B. Journal of Chemical Physics 2002, 116 (1), 209–217.

2.2 fix acks2/reaxff command

Accelerator Variants: [*acks2/reaxff/kk*](#)

2.2.1 Syntax

```
fix ID group-ID acks2/reaxff Nevery cutlo cuthi tolerance params args
```

- ID, group-ID are documented in [fix](#) command
 - acks2/reaxff = style name of this fix command
 - Nevery = perform ACKS2 every this many steps
 - cutlo,cuthi = lo and hi cutoff for Taper radius
 - tolerance = precision to which charges will be equilibrated
 - params = reaxff or a filename
 - one or more keywords or keyword/value pairs may be appended
- keyword = maxiter
maxiter N = limit the number of iterations to N

2.2.2 Examples

```
fix 1 all acks2/reaxff 1 0.0 10.0 1.0e-6 reaxff
fix 1 all acks2/reaxff 1 0.0 10.0 1.0e-6 param.acks2 maxiter 500
```

2.2.3 Description

Perform the atom-condensed Kohn–Sham DFT to second order (ACKS2) charge equilibration method as described in ([Verstraelen](#)). ACKS2 impedes unphysical long-range charge transfer sometimes seen with QEeq (e.g., for dissociation of molecules), at increased computational cost. It is typically used in conjunction with the ReaxFF force field model as implemented in the [pair_style reaxff](#) command, but it can be used with any potential in LAMMPS, so long as it defines and uses charges on each atom. For more technical details about the charge equilibration performed by fix acks2/reaxff, see the ([O'Hearn](#)) paper.

The ACKS2 method minimizes the electrostatic energy of the system by adjusting the partial charge on individual atoms based on interactions with their neighbors. It requires some parameters for each atom type. If the *params* setting above is the word “reaxff”, then these are extracted from the [pair_style reaxff](#) command and the ReaxFF force field file it reads in. If a file name is specified for *params*, then the parameters are taken from the specified file and the file must contain one line for each atom type. The latter form must be used when performing QEeq with a non-ReaxFF potential. The lines should be formatted as follows:

```
bond_softness
itype chi eta gamma bcut
```

where the first line is the global parameter *bond_softness*. The remaining 1 to Ntypes lines include *itype*, the atom type from 1 to Ntypes, *chi*, the electronegativity in eV, *eta*, the self-Coulomb potential in eV, *gamma*, the valence orbital exponent, and *bcut*, the bond cutoff distance. Note that these 4 quantities are also in the ReaxFF potential file, except that eta is defined here as twice the eta value in the ReaxFF file. Note that unlike the rest of LAMMPS, the units of this fix are hard-coded to be Å, eV, and electronic charge.

The optional *maxiter* keyword allows changing the max number of iterations in the linear solver. The default value is 200.

Note

In order to solve the self-consistent equations for electronegativity equalization, LAMMPS imposes the additional constraint that all the charges in the fix group must add up to zero. The initial charge assignments should also satisfy this constraint. LAMMPS will print a warning if that is not the case.

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

No information about this fix is written to *binary restart files*. This fix computes a global scalar (the number of iterations) for access by various *output commands*. No parameter of this fix can be used with the *start/stop* keywords of the *run* command.

This fix is invoked during *energy minimization*.

Styles with a *gpu*, *intel*, *kk*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed on the [Accelerator packages](#) page. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the GPU, INTEL, KOKKOS, OPENMP, and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the [Build package](#) page for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the *-suffix command-line switch* when you invoke LAMMPS, or you can use the *suffix* command in your input script.

See the [Accelerator packages](#) page for more instructions on how to use the accelerated styles effectively.

2.2.5 Restrictions

This fix is part of the REAXFF package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

This fix does not correctly handle interactions involving multiple periodic images of the same atom. Hence, it should not be used for periodic cell dimensions less than 10 Å.

This fix may be used in combination with *fix efield* and will apply the external electric field during charge equilibration, but there may be only one fix efield instance used, it may only use a constant electric field, and the electric field vector may only have components in non-periodic directions.

2.2.6 Related commands

pair_style reaxff, *fix qeq/reaxff*

2.2.7 Default

maxiter 200

(O'Hearn) O'Hearn, Alperen, Aktulga, SIAM J. Sci. Comput., 42(1), C1–C22 (2020).

(Verstraelen) Verstraelen, Ayers, Speybroeck, Waroquier, J. Chem. Phys. 138, 074108 (2013).

2.3 fix adapt command

2.3.1 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 *bond* or *angle* or *kspace* or *atom*

pair args = pstyle pparam I J v_name

pstyle = pair style name (e.g., lj/cut)

pparam = parameter to adapt over time

I,J = type pair(s) to set parameter for (integer or type label)

v_name = variable with name that calculates value of pparam

bond args = bstyle bparam I v_name

bstyle = bond style name (e.g., harmonic)

bparam = parameter to adapt over time

I = type bond to set parameter for (integer or type label)

v_name = variable with name that calculates value of bparam

angle args = astyle aparam I v_name

astyle = angle style name (e.g., harmonic)

aparam = parameter to adapt over time

I = type angle to set parameter for (integer or type label)

v_name = variable with name that calculates value of aparam

kspace arg = v_name

v_name = variable with name that calculates scale factor on *k*-space terms

atom args = atomparam v_name

atomparam = charge or diameter or diameter/disc = parameter to adapt over time

v_name = variable with name that calculates value of atomparam

- zero or more keyword/value pairs may be appended

- keyword = *scale* or *reset* or *mass*

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

mass value = no or yes
 no = mass is not altered by changes in diameter
 yes = mass is altered by changes in diameter

2.3.2 Examples

```
fix 1 all adapt 1 pair soft a 1 1 v_prefactor
fix 1 all adapt 1 pair soft a 2* 3 v_prefactor
fix 1 all adapt 1 pair lj/cut epsilon ** v_scale1 pair coul/cut scale 3 3 v_scale2 scale yes reset yes
fix 1 all adapt 10 atom diameter v_size

variable ramp_up equal "ramp(0.01,0.5)"
fix stretch all adapt 1 bond harmonic r0 1 v_ramp_up

labelmap atom 1 c1
fix 1 all adapt 1 pair soft a c1 c1 v_prefactor
```

2.3.3 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*, which is the default, then the value of the altered parameter 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).

Note that whether *scale* is *no* or *yes*, internally, the parameters themselves are actually altered by this fix. 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. If there are multiple sub-styles using the same pair style, then *pstyle* should be specified as “style: N ”, where N is which instance of the pair style you wish to adapt (e.g., the first or second). For example, *pstyle* could be specified as “soft” or “lubricate” or “lj/cut:1” or “lj/cut:2”. 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:

<i>born</i>	a,b,c	type pairs
<i>born/coul/long, born/coul/msm</i>	coulombic_cutoff	type global
<i>born/gauss</i>	biga0,biga1,r0	type pairs
<i>buck, buck/coul/cut</i>	a,c	type pairs
<i>buck/coul/long, buck/coul/msm</i>	a,c,coulombic_cutoff	type pairs
<i>buck/mdf</i>	a,c	type pairs
<i>coul/cut, coul/cut/global</i>	scale	type pairs
<i>coul/cut/soft</i>	lambda	type pairs
<i>coul/debye</i>	scale	type pairs
<i>coul/dsf</i>	coulombic_cutoff	type global
<i>coul/long, coul/msm</i>	coulombic_cutoff, scale	type pairs
<i>coul/long/soft</i>	scale, lambda, coulombic_cutoff	type pairs
<i>coul/slater/long</i>	scale	type pairs
<i>coul/streitz</i>	scale	type pairs
<i>eam, eam/alloy, eam/fs</i>	scale	type pairs
<i>gauss</i>	a	type pairs
<i>harmonic/cut</i>	k, cutoff	type pairs
<i>kim</i>	scale	type global
<i>lennard/mdf</i>	A,B	type pairs
<i>lj/class2</i>	epsilon,sigma	type pairs
<i>lj/class2/coul/cut, lj/class2/coul/long</i>	epsilon,sigma,coulombic_cutoff	type pairs
<i>lj/cut</i>	epsilon,sigma	type pairs
<i>lj/cut/coul/cut, lj/cut/coul/long, lj/cut/coul/msm</i>	epsilon,sigma,coulombic_cutoff	type pairs
<i>lj/cut/coul/cut/soft, lj/cut/coul/long/soft</i>	epsilon,sigma,lambda,coulombic_cutoff	type pairs
<i>lj/cut/coul/dsf</i>	cutoff	type global
<i>lj/cut/tip4p/cut</i>	epsilon,sigma,coulombic_cutoff	type pairs
<i>lj/cut/soft</i>	epsilon,sigma,lambda	type pairs
<i>lj/expand</i>	epsilon,sigma,delta	type pairs
<i>lj/mdf</i>	epsilon,sigma	type pairs
<i>lj/sf/dipole/sf</i>	epsilon,sigma,lambda	type pairs
<i>lubricate</i>	mu	global
<i>meam</i>	scale	type pairs
<i>mie/cut</i>	epsilon,sigma,gamma_repulsive,gamma_attractive	type pairs
<i>morse, morse/smooth/linear</i>	D0,R0,alpha	type pairs
<i>morse/soft</i>	D0,R0,alpha,lambda	type pairs
<i>nm/cut</i>	E0,R0,m,n	type pairs
<i>nm/cut/coul/cut, nm/cut/coul/long</i>	E0,R0,m,n,coulombic_cutoff	type pairs
<i>pace, pace/extrapolation</i>	scale	type pairs
<i>quip</i>	scale	type global
<i>snap</i>	scale	type pairs
<i>spin/dmi</i>	coulombic_cutoff	type global
<i>spin/exchange</i>	coulombic_cutoff	type global
<i>spin/magelec</i>	coulombic_cutoff	type global
<i>spin/neel</i>	coulombic_cutoff	type global
<i>soft</i>	a	type pairs
<i>table</i>	table_cutoff	type pairs
<i>ufm</i>	epsilon,sigma,lambda	type pairs
<i>wf/cut</i>	epsilon,sigma,nu,mu	type pairs

 **Note**

It is easy to add new pairwise potentials and their parameters to this list. All it typically takes is adding an extract() method to the pair_*.cpp file associated with the potential.

Some parameters are global settings for the pair style (e.g., the viscosity setting “mu” for *pair_style lubricate*). Other parameters apply to atom type pairs within the pair style (e.g., the prefactor *a* for *pair_style soft*).

Note that for many of the potentials, the parameter that can be varied is effectively a prefactor on the entire energy expression for the potential (e.g., the lj/cut epsilon). The parameters listed as “scale” are exactly that, since the energy expression for the *coul/cut* potential (for example) has no labeled prefactor in its formula. To apply an effective prefactor to some potentials, multiple parameters need to be altered. For example, the *Buckingham potential* needs both the *A* and *C* terms altered together. To scale the Buckingham potential, you should thus list the pair style twice, once for *A* and once for *C*.

If a type pair parameter is specified, the *I* and *J* settings should be specified to indicate which type pairs to apply it to. If a global parameter is specified, the *I* and *J* settings still need to be specified, but are ignored.

Similar to the *pair_coeff command*, *I* and *J* can be specified in one of several ways. Explicit numeric values can be used for each, as in the first example above. Or, one or both of the types in the *I,J* pair can be a *type label*. LAMMPS sets the coefficients for the symmetric *J,I* interaction to the same values.

A wild-card asterisk can be used in place of or in conjunction with the *I,J* arguments to set the coefficients for multiple pairs of atom types. This takes the form “*” or “**n*” or “*m**” or “*m***n*”. If *N* is 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 *m* to *N* (inclusive). A middle asterisk means all types from *m* to *n* (inclusive). For the asterisk syntax, note that only type pairs with $I \leq J$ are considered; if asterisks imply type pairs where $J < I$, they are ignored.

IMPORTANT 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 *bond* keyword uses the specified variable to change the value of a bond coefficient over time, very similar to how the *pair* keyword operates. The only difference is that now a bond coefficient for a given bond type is adapted.

A wild-card asterisk can be used in place of or in conjunction with the bond type argument to set the coefficients for multiple bond types. This takes the form “*” or “**n*” or “*m**” or “*m***n*”. If *N* is 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 *m* to *N* (inclusive). A middle asterisk means all types from *m* to *n* (inclusive).

Currently *bond* does not support *bond_style hybrid* nor *bond_style hybrid/overlay* as bond styles. The bond styles that currently work with *fix_adapt* are

<i>class2</i>	r0	type bonds
<i>fene</i>	k,r0	type bonds
<i>fene/nm</i>	k,r0	type bonds
<i>gromos</i>	k,r0	type bonds
<i>harmonic</i>	k,r0	type bonds
<i>morse</i>	r0	type bonds
<i>nonlinear</i>	epsilon,r0	type bonds

Added in version 4May2022.

The *angle* keyword uses the specified variable to change the value of an angle coefficient over time, very similar to how the *pair* keyword operates. The only difference is that now an angle coefficient for a given angle type is adapted.

A wild-card asterisk can be used in place of or in conjunction with the angle type argument to set the coefficients for multiple angle types. This takes the form “*” or “*n” or “m*” or “m*n”. If *N* is the number of angle types, then an asterisk with no numeric values means all types from 1 to *N*. A leading asterisk means all types from 1 to *n* (inclusive). A trailing asterisk means all types from *m* to *N* (inclusive). A middle asterisk means all types from *m* to *n* (inclusive).

Currently *angle* does not support angle_style hybrid nor angle_style hybrid/overlay as angle styles. The angle styles that currently work with fix_adapt are

<i>harmonic</i>	k,theta0	type angles
<i>cosine</i>	k	type angles

Note that internally, theta0 is stored in radians, so the variable this fix uses to reset theta0 needs to generate values in radians.

The *kspace* keyword used the specified variable as a scale factor on the energy, forces, virial calculated by whatever *k*-space solver is defined by the *kspace_style* command. If the variable has a value of 1.0, then the solver is unaltered.

The *kspace* keyword works this way whether the *scale* keyword is set to *no* or *yes*.

The *atom* keyword enables various atom properties to be changed. The *aparam* argument is the name of the parameter to change. This is the current list of atom parameters that can be varied by this fix:

- charge = charge on particle
- diameter or diameter/disc = 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, or scale 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, by default, also changed when the diameter changes. The mass is set from the particle volume for 3d systems (density is assumed to stay constant). For 2d, the default is for LAMMPS to model particles with a radius attribute as spheres. However, if the atom parameter is *diameter/disc*, then the mass is set from the particle area (the density is assumed to be in mass/distance² units). The mass of the particle may also be kept constant if the *mass* keyword is set to *no*. This can be useful to account for diameter changes that do not involve mass changes (e.g., thermal expansion).

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
```

This fix can be used in long simulations which are restarted one or more times to continuously adapt simulation parameters, but it must be done carefully. There are two issues to consider. The first is how to adapt the parameters in a continuous manner from one simulation to the next. The second is how, if desired, to reset the parameters to their original values at the end of the last restarted run.

Note that all the parameters changed by this fix are written into a restart file in their current changed state. A new restarted simulation does not know the original time=0 values, unless the input script explicitly resets the parameters (after the restart file is read) to the original values.

Also note that the time-dependent variable(s) used in the restart script should typically be written as a function of time elapsed since the original simulation began.

With this in mind, if the *scale* keyword is set to *no* (the default) in a restarted simulation, original parameters are not needed. The adapted parameters should seamlessly continue their variation relative to the preceding simulation.

If the *scale* keyword is set to *yes*, then the input script should typically reset the parameters being adapted to their original values, so that the scaling formula specified by the variable will operate correctly. An exception is if the *atom* keyword is being used with *scale yes*. In this case, information is added to the restart file so that per-atom properties in the new run will automatically be scaled relative to their original values. This will only work if the fix *adapt* command specified in the restart script has the same ID as the one used in the original script.

In a restarted run, if the *reset* keyword is set to *yes*, and the run ends in this script (as opposed to just writing more restart files), parameters will be restored to the values they were at the beginning of the run command in the restart script, which as explained above, may or may not be the original values of the parameters. Again, an exception is if the *atom* keyword is being used with *reset yes* (in all the runs). In that case, the original per-atom parameters are stored in the restart file, and will be restored when the restarted run finally completes.

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

If the *atom* keyword is used and the *scale* or *reset* keyword is set to *yes*, then this fix writes information to a restart file so that in a restarted run scaling can continue in a seamless manner and/or the per-atom values can be restored, as explained above.

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*.

For *rRESPA time integration*, this fix changes parameters on the outermost rRESPA level.

2.3.5 Restrictions

none

2.3.6 Related commands

compute ti, fix adapt/fep

2.3.7 Default

The option defaults are scale = no, reset = no, mass = yes.

2.4 fix adapt/fep command

2.4.1 Syntax

```
fix ID group-ID adapt/fep N attribute args ... keyword value ...
```

- ID, group-ID are documented in [fix](#) command
- adapt/fep = style name of this fix command
- N = adapt simulation settings every this many timesteps
- one or more attribute/arg pairs may be appended
- attribute = *pair* or *kspace* or *atom*

pair args = pstyle pparam I J v_name
 pstyle = pair style name (e.g., lj/cut)
 pparam = parameter to adapt over time
 I,J = type pair(s) to set parameter for (integer or type label)
 v_name = variable with name that calculates value of pparam
 kspace arg = v_name
 v_name = variable with name that calculates scale factor on K-space terms
 atom args = aparam v_name
 aparam = parameter to adapt over time
 I = type(s) to set parameter for (integer or type label)
 v_name = variable with name that calculates value of aparam

- zero or more keyword/value pairs may be appended
- keyword = *scale* or *reset* or *after*

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
 after value = no or yes
 no = parameters are adapted at timestep N
 yes = parameters are adapted one timestep after N

2.4.2 Examples

```

fix 1 all adapt/fep 1 pair soft a 1 1 v_prefactor
fix 1 all adapt/fep 1 pair soft a 2* 3 v_prefactor
fix 1 all adapt/fep 1 pair lj/cut epsilon ** v_scale1 coul/cut scale 3 3 v_scale2 scale yes reset yes
fix 1 all adapt/fep 10 atom diameter 1 v_size

labelmap atom 1 c1
fix 1 all adapt/fep 1 pair soft a c1 c1 v_prefactor

```

Example input scripts available: examples/PACKAGES/fep

2.4.3 Description

Change or adapt one or more specific simulation attributes or settings over time as a simulation runs.

This is an enhanced version of the [fix adapt](#) command with two differences:

- It is possible to modify the charges of chosen atom types only, instead of scaling all the charges in the system.
- There is a new option *after* for better compatibility with [fix ave/time](#).

This version is suited for free energy calculations using [compute ti](#) or [compute fep](#).

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.

If the *after* keyword is set to *yes*, then the parameters are changed one timestep after the multiple of *N*. In this manner, if a fix such as “fix ave/time” is used to calculate averages at every *N* timesteps, all the contributions to the average will be obtained with the same values of the parameters.

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:

<i>born</i>	a,b,c	type pairs
<i>born/gauss</i>	biga0,biga1,r0	type pairs
<i>buck, buck/coul/cut, buck/coul/long, buck/coul/msm</i>	a,c	type pairs

continues on next page

Table 2 – continued from previous page

<i>buck/mdf</i>	a,c	type pairs
<i>coul/cut, coul/cut/global</i>	scale	type pairs
<i>coul/cut/soft</i>	lambda	type pairs
<i>coul/debye</i>	scale	type pairs
<i>coul/long, coul/msm</i>	scale	type pairs
<i>coul/long/soft</i>	scale, lambda	type pairs
<i>coul/slater/long</i>	scale	type pairs
<i>coul/streitz</i>	scale	type pairs
<i>eam, eam/alloy, eam/fs</i>	scale	type pairs
<i>harmonic/cut</i>	k	type pairs
<i>gauss</i>	a	type pairs
<i>lennard/mdf</i>	a,b	type pairs
<i>lj/class2</i>	epsilon,sigma	type pairs
<i>lj/class2/coul/cut, lj/class2/coul/long</i>	epsilon,sigma	type pairs
<i>lj/cut</i>	epsilon,sigma	type pairs
<i>lj/cut/soft</i>	epsilon,sigma,lambda	type pairs
<i>lj/cut/coul/cut, lj/cut/coul/long, lj/cut/coul/msm</i>	epsilon,sigma	type pairs
<i>lj/cut/coul/cut/soft, lj/cut/coul/long/soft</i>	epsilon,sigma,lambda	type pairs
<i>lj/cut/tip4p/cut, lj/cut/tip4p/long</i>	epsilon,sigma	type pairs
<i>lj/cut/tip4p/long/soft</i>	epsilon,sigma,lambda	type pairs
<i>lj/expand</i>	epsilon,sigma,delta	type pairs
<i>lj/mdf</i>	epsilon,sigma	type pairs
<i>lj/sf/dipole/sf</i>	epsilon,sigma,scale	type pairs
<i>meam</i>	scale	type pairs
<i>mie/cut</i>	epsilon,sigma,gamR,gamA	type pairs
<i>morse, morse/smooth/linear</i>	d0,r0,alpha	type pairs
<i>morse/soft</i>	d0,r0,alpha,lambda	type pairs
<i>nm/cut</i>	e0,r0,nn,mm	type pairs
<i>nm/cut/coul/cut, nm/cut/coul/long</i>	e0,r0,nn,mm	type pairs
<i>pace, pace/extrapolation</i>	scale	type pairs
<i>snap</i>	scale	type pairs
<i>soft</i>	a	type pairs
<i>ufm</i>	epsilon,sigma,scale	type pairs
<i>wf/cut</i>	epsilon,sigma,nu,mu	type pairs

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.

Note that for many of the potentials, the parameter that can be varied is effectively a prefactor on the entire energy expression for the potential (e.g., the *lj/cut epsilon*). The parameters listed as “scale” are exactly that, since the energy expression for the *coul/cut* potential (for example) has no labeled prefactor in its formula. To apply an effective prefactor to some potentials, multiple parameters need to be altered. For example, the *Buckingham potential* needs both the A and C terms altered together. To scale the Buckingham potential, you should thus list the pair style twice, once for A and once for C.

If a type pair parameter is specified, the *I* and *J* settings should be specified to indicate which type pairs to apply it to. If a global parameter is specified, the *I* and *J* settings still need to be specified, but are ignored.

Similar to the *pair_coeff command*, *I* and *J* can be specified in one of several ways. Explicit numeric values can be used for each, as in the first example above. Or, one or both of the types in the I,J pair can be a *type label*. LAMMPS

sets the coefficients for the symmetric J, I interaction to the same values.

A wild-card asterisk can be used in place of or in conjunction with the I, J arguments to set the coefficients for multiple pairs of atom types. This takes the form “*” or “*n” or “m*” or “m*n”. If N is 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 m to N (inclusive). A middle asterisk means all types from m to n (inclusive). For the asterisk syntax, note that only type pairs with $I \leq J$ are considered; if asterisks imply type pairs where $J < I$, they are ignored.

IMPORTANT NOTE: If [pair_style hybrid or hybrid/overlay](#) is being used, then the *pstyle* will be a sub-style name. You must specify I, J arguments that correspond to type pair values defined (via the [pair_coeff](#) command) for that sub-style.

The *v_name* argument for keyword *pair* is the name of an [equal-style variable](#) which will be evaluated each time this fix is invoked to set the parameter to a new value. It should be specified as *v_name*, where *name* is the variable name. Equal-style variables can specify formulas with various mathematical functions, and include [thermo_style](#) command keywords for the simulation box parameters and timestep and elapsed time. Thus it is easy to specify parameters that change as a function of time or span consecutive runs in a continuous fashion. For the latter, see the *start* and *stop* keywords of the [run](#) command and the *elaplong* keyword of [thermo_style custom](#) for details.

For example, these commands would change the prefactor coefficient of the [pair_style soft](#) potential from 10.0 to 30.0 in a linear fashion over the course of a simulation:

```
variable prefactor equal ramp(10,30)
fix 1 all adapt 1 pair soft a * * v_prefactor
```

The *kspace* keyword used the specified variable as a scale factor on the energy, forces, virial calculated by whatever *k*-space solver is defined by the [kspace_style](#) command. If the variable has a value of 1.0, then the solver is unaltered.

The *kspace* keyword works this way whether the *scale* keyword is set to *no* or *yes*.

The *atom* keyword enables various atom properties to be changed. The *aparam* argument is the name of the parameter to change. This is the current list of atom parameters that can be varied by this fix:

- charge = charge on particle
- diameter = diameter of particle

The *I* argument indicates which atom types are affected. A wild-card asterisk can be used in place of or in conjunction with the *I* argument to set the coefficients for multiple atom types.

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
```

For [rRESPA time integration](#), this fix changes parameters on the outermost rRESPA level.

2.4.4 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](#).

2.4.5 Restrictions

The keyword “scale yes” is not supported for scaling per-atom parameters diameter and change. You can use [fix adapt](#) for those.

2.4.6 Related commands

[compute fep](#), [fix adapt](#), [compute ti](#), [pair_style */soft](#)

2.4.7 Default

The option defaults are scale = no, reset = no, after = no.

2.5 fix add/heat command

2.5.1 Syntax

`fix ID group-ID add/heat style args keyword values ...`

- ID, group-ID are documented in [fix](#) command
- add/heat = style name of this fix command
- style = *constant* or *linear* or *quartic*
 - constant args = rate
rate = rate of heat flow (energy/time units)
 - linear args = Ttarget k
Ttarget = target temperature (temperature units)
k = prefactor (energy/(time*temperature) units)
 - quartic args = Ttarget k
Ttarget = target temperature (temperature units)
k = prefactor (energy/(time*temperature^4) units)
- zero or more keyword/value pairs may be appended to args
- keyword = *overwrite*
 - overwrite value = yes or no
yes = sets current heat flow of particle
no = adds to current heat flow of particle

2.5.2 Examples

```
fix 1 all add/heat constant v_heat
fix 1 all add/heat linear 10.0 1.0 overwrite yes
```

2.5.3 Description

This fix adds heat to particles with the temperature attribute every timestep at a given rate. Note that this is an internal temperature of a particle intended for use with non-atomistic models like the discrete element method.

For the *constant* style, heat is added at the specified rate. For the *linear* style, heat is added at a rate of $k(T_{target} - T)$ where k is the specified prefactor, T_{target} is the specified target temperature, and T is the temperature of the atom. This may be more representative of a conductive process. For the *quartic* style, heat is added at a rate of $k(T_{target}^4 - T^4)$, akin to radiative heat transfer.

The rate or temperature 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 each time step, and its value will be used to determine the rate of heat added.

Equal-style variables can specify formulas with various mathematical functions and include *thermo_style* command keywords for the simulation box parameters, time step, and elapsed time to specify time-dependent heating.

Atom-style variables can specify the same formulas as equal-style variables but can also include per-atom values, such as atom coordinates to specify spatially-dependent heating.

If the *overwrite* keyword is set to *yes*, this fix will set the total heat flow on a particle every timestep, overwriting contributions from pair styles or other fixes. If *overwrite* is *no*, this fix will add heat on top of other contributions.

2.5.4 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*.

2.5.5 Restrictions

This pair style is part of the GRANULAR package. It is only enabled if LAMMPS was built with that package. See the *Build package* page for more info.

This fix requires that atoms store temperature and heat flow as defined by the *fix property/atom* command or included in certain atom styles, such as `atom_style rheo/thermal`.

2.5.6 Related commands

fix heat/flow, fix property/atom, fix rheo/thermal

2.5.7 Default

The default for the *overwrite* keyword is *no*

2.6 fix addforce command

2.6.1 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 = *every* or *region* or *energy*

every value = *Nevery*

Nevery = add force every this many time steps

region value = *region-ID*

region-ID = ID of region atoms must be in to have added force

energy value = *v_name*

v_name = variable with name that calculates the potential energy of each atom in the added

force field

2.6.2 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
```

2.6.3 Description

Add (f_x, f_y, f_z) to the corresponding component of the 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 three quantities defining the force components, namely f_x , f_y , and f_z , 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 each time step, and its value(s) will be used to determine the force component(s).

Equal-style variables can specify formulas with various mathematical functions and include [thermo_style](#) command keywords for the simulation box parameters, time step, 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 *every* keyword is used, the *Nevery* setting determines how often the forces are applied. The default value is 1, for every time step.

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 $\vec{F} = (f_x, f_y, f_z)$, with all components defined as numeric constants and not as variables. This is because LAMMPS can compute the energy for each atom directly as

$$E = -\vec{x} \cdot \vec{F} = -(x f_x + y f_y + z f_z),$$

so that $-\nabla E = \vec{F}$.

The *energy* keyword is optional if the added force is defined with one or more variables, and if you are performing dynamics via the *run* command. If the keyword is not used, LAMMPS will set the energy to 0.0, which is typically fine for dynamics.

The *energy* keyword is required if the added force is defined with one or more variables, and you are performing energy minimization via the “minimize” command. The keyword specifies the name of an atom-style *variable* which is used to compute the energy of each atom as function of its position. Like variables used for f_x, f_y, f_z , 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 ensure that the formula defined for the atom-style *variable* is consistent with the force variable formulas (i.e., that $-\nabla E = \vec{F}$). For example, if the force were a spring-like, $\vec{F} = -k\vec{x}$, then the energy formula should be $E = \frac{1}{2}kx^2$. If you do not do this correctly, the minimization will not converge properly.

Styles with a *gpu*, *intel*, *kk*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed on the *Accelerator packages* page. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the GPU, INTEL, KOKKOS, OPENMP, and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the *Build package* page for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the *-suffix command-line switch* when you invoke LAMMPS, or you can use the *suffix* command in your input script.

See the *Accelerator packages* page for more instructions on how to use the accelerated styles effectively.

2.6.4 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 global potential energy of the system as part of *thermodynamic output*. The default setting for this fix is *fix_modify energy no*. Note that this energy 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).

The *fix_modify virial* option is supported by this fix to add the contribution due to the added forces on atoms to both the global pressure and per-atom stress of the system via the *compute pressure* and *compute stress/atom* commands. The former can be accessed by *thermodynamic output*. The default setting for this fix is *fix_modify virial no*.

The *fix_modify respa* option is supported by this fix. This allows to set at which level of the *r-RESPA* integrator the fix is adding its forces. Default is the outermost level.

This fix computes a global scalar and a global three-vector of forces, which can be accessed by various *output commands*. The scalar is the potential energy discussed above. The vector is the total force on the group of atoms before the forces on individual atoms are changed by the fix. The scalar and vector values calculated by this fix are “extensive”.

No parameter of this fix can be used with the *start/stop* keywords of the *run* command.

The forces due to this fix are imposed during an energy minimization, invoked by the *minimize* command. You should not specify force components with a variable that has time-dependence for use with a minimizer, since the minimizer increments the time step as the iteration count during the minimization.

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.

2.6.5 Restrictions

none

2.6.6 Related commands

fix setforce, *fix aveforce*

2.6.7 Default

The option default for the every keyword is every = 1.

2.7 fix addtorque command

2.7.1 Syntax

```
fix ID group-ID addtorque Tx Ty Tz
```

- ID, group-ID are documented in [fix](#) command
- addtorque = style name of this fix command
- Tx,Ty,Tz = torque component values (torque units)
- any of Tx,Ty,Tz can be a variable (see below)

2.7.2 Examples

```
fix kick bead addtorque 2.0 3.0 5.0
fix kick bead addtorque 0.0 0.0 v_oscillate
```

2.7.3 Description

Add a set of forces to each atom in the group such that:

- the components of the total torque applied on the group (around its center of mass) are T_x , T_y , and T_z
- the group would move as a rigid body in the absence of other forces.

This command can be used to drive a group of atoms into rotation.

Any of the three quantities defining the torque components can be specified as an equal-style [variable](#), namely T_x , T_y , T_z . If the value is a variable, it should be specified as v_name , where $name$ is the variable name. In this case, the variable will be evaluated each timestep, and its value used to determine the torque component.

Equal-style variables can specify formulas with various mathematical functions, and include [thermo_style](#) command keywords for the simulation box parameters and timestep and elapsed time. Thus it is easy to specify a time-dependent torque.

2.7.4 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 torques to the global potential energy of the system as part of [thermodynamic output](#). The default setting for this fix is [fix_modify energy no](#). Note that this is a fictitious quantity but is needed so that the [minimize](#) command can include the forces added by this fix in a consistent manner (i.e., there is a decrease in potential energy when atoms move in the direction of the added forces).

The [fix_modify respa](#) option is supported by this fix. This allows to set at which level of the [r-RESPA](#) integrator the fix is adding its torque. Default is the outermost level.

This fix computes a global scalar and a global 3-vector, which can be accessed by various [output commands](#). The scalar is the potential energy discussed above. The vector is the total torque on the group of atoms before the forces on individual atoms are changed by the fix. The scalar and vector values calculated by this fix are “extensive”.

No parameter of this fix can be used with the *start/stop* keywords of the [run](#) command.

The forces due to this fix are imposed during an energy minimization, invoked by the [minimize](#) command.

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.

Note

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.

2.7.5 Restrictions

This fix is part of the EXTRA-FIX package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

2.7.6 Related commands

[fix addforce](#)

2.7.7 Default

none

2.8 fix alchemy command

2.8.1 Syntax

```
fix ID group-ID alchemy v_name
```

- ID, group-ID are documented in [fix](#) command
- alchemy = style name of this fix command
- v_name = variable with name that determines the λ_R value

2.8.2 Examples

```
fix trans all alchemy v_ramp
```

2.8.3 Description

Added in version 28Mar2023.

This fix command enables an “alchemical transformation” to be performed between two systems, whereby one system slowly transforms into the other over the course of a molecular dynamics run. This is useful for measuring thermodynamic differences between two different systems. It also allows transformations that are not easily possible with the [pair style hybrid/scaled](#), [fix adapt](#) or [fix adapt/sep](#) commands.

Example inputs are included in the examples/PACKAGES/alchemy directory for (a) transforming a pure copper system into a copper/aluminum bronze alloy and (b) transforming two water molecules in a box of water into a hydronium and a hydroxyl ion.

The two systems must be defined as [separate replica](#) and run in separate partitions of processors using the [-partition](#) command-line switch. Exactly two partitions must be specified, and each partition must use the same number of processors and the same domain decomposition.

Because the forces applied to the atoms are the same mix of the forces from each partition and the simulation starts with the same atom positions across both partitions, they will generate the same trajectory of coordinates for each atom, and the same simulation box size and shape. The latter two conditions are *enforced* by this fix; it exchanges coordinates and box information between the replicas. This is not strictly required, but since MD simulations are an example of a chaotic system, even the tiniest random difference will eventually grow exponentially into an unwanted divergence.

Otherwise, the properties of each atom (type, charge, bond and angle partners, etc.), as well as energy and forces between interacting atoms (pair, bond, angle styles, etc.) can be different in the two systems.

This can be initialized in the same input script by using commands which only apply to one or the other replica. The example scripts use a world-style [variable](#) command along with [if/then/else](#) commands for this purpose. The [partition](#) command can also be used.

```
create_box 2 box
create_atoms 1 box
pair_style eam/alloy
pair_coeff * * AlCu.eam.alloy Cu Al

# replace 5% of copper with aluminum on the second partition only

variable name world pure alloy
if "{$name} == alloy" then &
  "set type 1 type/fraction 2 0.05 6745234"
```

Both replicas must define an instance of this fix, but with a different *v_name* variable. The named variable must be an equal-style or equivalent [variable](#). The two variables should be defined so that one ramps *down* from 1.0 to 0.0 for the *first* replica (*R=0*) and the other ramps *up* from 0.0 to 1.0 for the *second* replica (*R=1*). A simple way is to do this linearly, which can be done using the ramp() function of the [variable](#) command. You could also define a variable which returns a value between 0.0 and 1.0 as a non-linear function of the timestep. Here is a linear example:

```
partition yes 1 variable ramp equal ramp(1.0,0.0)
partition yes 2 variable ramp equal ramp(0.0,1.0)
fix 2 all alchemy v_ramp
```

Note

For an alchemical transformation, the two variables should sum to exactly 1.0 at any timestep. LAMMPS does *NOT* check that this is the case.

If you use the ramp() function to define the two variables, this fix can easily be used across successive runs in the same input script by ensuring each instance of the [run](#) command specifies the appropriate *start* or *stop* options.

At each timestep of an MD run, the two instances of this fix evaluate their respective variables as a λ_R factor, where $R = 0$ or 1 for each replica. The forces used by each system for the propagation of their atoms is set to the sum of the forces for the two systems, each scaled by their respective λ_R factor. Thus, during the MD run, the system will transform incrementally from the first system to the second system.

Note

As mentioned above, the coordinates of the atoms and box size/shape must be exactly the same in the two replicas. Therefore, it is generally not a good idea to initialize the two replicas by reading different data files or creating them individually from scratch. Rather, a single system should be initialized and then desired modifications applied to the system to either replica. If your input script somehow induces the two systems to become different (e.g. by performing [atom_modify sort](#) differently, or by adding or depositing a different number of atoms), then LAMMPS will detect the mismatch and generate an error. This is done by ensuring that each step the number and ordering of atoms is identical within each pair of processors in the two replicas.

2.8.4 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 stores a global scalar (the current value of λ_R) and a global vector of length 3 which contains the potential energy of the first partition, the second partition and the combined value, respectively. The global scalar is unitless and “intensive”, the vector is in *energy units* and “extensive”. These values can be used by any command that uses a global value from a fix as input. See the [output howto](#) page for an overview of LAMMPS output options.

This fix is not invoked during *energy minimization*.

2.8.5 Restrictions

This fix is part of the REPLICA package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

There may be only one instance of this fix in use at a time within each replica.

2.8.6 Related commands

[compute pressure/alchemy](#) command, [fix adapt](#) command, [fix adapt/sep](#) command, [pair_style hybrid/scaled](#) command.

2.8.7 Default

none

2.9 fix amoeba/bitorsion command

2.9.1 Syntax

```
fix ID group-ID amoeba/bitorsion filename
```

- ID, group-ID are documented in [fix](#) command
- amoeba/bitorsion = style name of this fix command
- filename = force-field file with AMOEBA bitorsion coefficients

2.9.2 Examples

```
fix      bit all amoeba/bitorsion bitorsion.ubiquitin.data
read_data   proteinX.data fix bit bitorsions BiTorsions
fix_modify  bit energy yes
```

2.9.3 Description

This command enables 5-body torsion/torsion interactions to be added to simulations which use the AMOEBA and HIPPO force fields. It matches how the Tinker MD code computes its torsion/torsion interactions for the AMOEBA and HIPPO force fields. See the [Howto amoeba](#) doc page for more information about the implementation of AMOEBA and HIPPO in LAMMPS.

Bitorsion interactions add additional potential energy contributions to pairs of overlapping phi-psi dihedrals of amino-acids, which are important to properly represent their conformational behavior. Each bitorsion interaction is thus defined for a 5-tuple of atoms $IJKLM$ with bonds between successive atoms in the list, i.e. two overlapping dihedral interactions for atoms $IJKL$ and $JKLM$.

The examples/amoeba directory has a sample input script and data file for ubiquitin, which illustrates use of the fix amoeba/bitorsion command.

As in the example above, this fix should be used before reading a data file that contains a listing of bitorsion interactions. The *filename* specified should contain the bitorsion parameters for the AMOEBA or HIPPO force field.

The data file read by the [read_data](#) command must contain the topology of all the bitorsion interactions, similar to the topology data for bonds, angles, dihedrals, etc. Specifically it should have a line like this in its header section:

```
N bitorsions
```

where N is the number of bitorsion 5-body interactions. It should also have a section in the body of the data file like this with N lines:

BiTorsions

1	1	8	10	12	18	20
2	5	18	20	22	25	27
[...]						
N	3	314	315	317	318	330

The first column is an index from 1 to N to enumerate the bitorsion 5-atom tuples; it is ignored by LAMMPS. The second column is the *type* of the interaction; it is an index into the bitorsion force field file. The remaining 5 columns are the atom IDs of the atoms (in order) for the 5-tuple $JKLM$, as described above.

Note that the *bitorsions* and *BiTorsions* keywords for the header and body sections match those specified in the *read_data* command following the data file name.

The data file should be generated by using the tools/tinker/tinker2lmp.py conversion script which creates a LAMMPS data file from Tinker input files, including its PRM file which contains the parameters necessary for computing bitorsion interactions. The script must be invoked with the optional “-bitorsion” flag to do this; see the example for the ubiquitin system in the tools/tinker/README file. The same conversion script also creates the file of bitorsion coefficient data which is read by this command.

The potential energy associated with bitorsion interactions can be output as described below. It can also be included in the total potential energy of the system, as output by the *thermo_style* command, if the *fix_modify energy* command is used, as in the example above. See the note below about how to include the bitorsion energy when performing an *energy minimization*.

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

This fix writes the list of bitorsion interactions to *binary restart files*. See the *read_restart* command for info on how to re-specify a fix in an input script that reads a restart file, so that the operation of the fix continues in an uninterrupted fashion.

The *fix_modify energy* option is supported by this fix to add the potential energy of the bitorsion interactions to both the global potential energy and per-atom potential energies of the system as part of *thermodynamic output* or output by the *compute pe/atom* command. The default setting for this fix is *fix_modify energy yes*.

The *fix_modify virial* option is supported by this fix to add the contribution due to the bitorsion interactions to both the global pressure and per-atom stress of the system via the *compute pressure* and *compute stress/atom* commands. The former can be accessed by *thermodynamic output*. The default setting for this fix is *fix_modify virial yes*.

This fix computes a global scalar which can be accessed by various *output commands*. The scalar is the potential energy discussed above. The scalar value calculated by this fix is “extensive”.

No parameter of this fix can be used with the *start/stop* keywords of the *run* command.

The forces due to this fix are imposed during an energy minimization, invoked by the *minimize* command.

The *fix_modify respa* option is supported by this fix. This allows to set at which level of the *r-RESPA* integrator the fix is adding its forces. Default is the outermost level.

Note

For energy minimization, if you want the potential energy associated with the bitorsion terms forces to be included in the total potential energy of the system (the quantity being minimized), you MUST not disable the *fix_modify energy* option for this fix.

2.9.5 Restrictions

To function as expected this fix command must be issued *before* a `read_data` command but *after* a `read_restart` command.

This fix can only be used if LAMMPS was built with the AMOEBA package. See the [Build package](#) page for more info.

2.9.6 Related commands

`fix_modify`, `read_data`

2.9.7 Default

none

2.10 fix amoeba/pitorsion command

2.10.1 Syntax

```
fix ID group-ID amoeba/pitorsion
```

- ID, group-ID are documented in `fix` command
- amoeba/pitorsion = style name of this fix command

2.10.2 Examples

```
fix      pit all amoeba/pitorsion
read_data  proteinX.data fix pit "pitorsion types" "PiTorsion Coeffs" &
           fix pit pitorsions PiTorsions
fix_modify  pit energy yes
```

2.10.3 Description

This command enables 6-body pitorsion interactions to be added to simulations which use the AMOEBA and HIPPO force fields. It matches how the Tinker MD code computes its pitorsion interactions for the AMOEBA and HIPPO force fields. See the [Howto amoeba](#) doc page for more information about the implementation of AMOEBA and HIPPO in LAMMPS.

Pitorsion interactions add additional potential energy contributions to 6-tuples of atoms $IJKLMN$ that have a bond between atoms K and L , where both K and L are additionally bonded to exactly two other atoms. Namely, K is also bonded to I and J , and L is also bonded to M and N .

The examples/amoeba directory has a sample input script and data file for ubiquitin, which illustrates use of the fix amoeba/pitorsion command.

As in the example above, this fix should be used before reading a data file that contains a listing of pitorsion interactions.

The data file read by the `read_data` command must contain the topology of all the pitorsion interactions, similar to the topology data for bonds, angles, dihedrals, etc. Specifically, it should have two lines like these in its header section:

M pitorsion types
N pitorsions

where N is the number of pitorsion 6-body interactions and M is the number of pitorsion types. It should also have two sections in the body of the data file like these with M and N lines each:

PiTorsion Coeffs

1	6.85
2	10.2
[...]	
M	6.85

PiTorsions

1	1	2	4	3	20	21	24
2	5	21	23	22	37	38	41
[...]							
N	7	27	29	28	30	35	36

For PiTorsion Coeffs, the first column is an index from 1 to M to enumerate the pitorsion types. The second column is the single prefactor coefficient needed for each type.

For PiTorsions, the first column is an index from 1 to N to enumerate the pitorsion 6-atom tuples; it is ignored by LAMMPS. The second column is the “type” of the interaction; it is an index into the PiTorsion Coeffs. The remaining 6 columns are the atom IDs of the atoms (in order) for the 6-tuple $IJKLMN$, as described above.

Note that the *pitorsion types* and *pitorsions* and *PiTorsion Coeffs* and *PiTorsions* keywords for the header and body sections of the data file match those specified in the [read_data](#) command following the data file name.

The data file should be generated by using the tools/tinker/tinker2lmp.py conversion script which creates a LAMMPS data file from Tinker input files, including its PRM file which contains the parameters necessary for computing pitorsion interactions.

The potential energy associated with pitorsion interactions can be output as described below. It can also be included in the total potential energy of the system, as output by the [thermo_style](#) command, if the [fix_modify energy](#) command is used, as in the example above. See the note below about how to include the pitorsion energy when performing an [energy minimization](#).

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

This fix writes the list of pitorsion interactions to *binary restart files*. See the [read_restart](#) command for info on how to re-specify a fix in an input script that reads a restart file, so that the operation of the fix continues in an uninterrupted fashion.

The [fix_modify energy](#) option is supported by this fix to add the potential energy of the pitorsion interactions to both the global potential energy and peratom potential energies of the system as part of [thermodynamic output](#) or output by the [compute pe/atom](#) command. The default setting for this fix is [fix_modify energy yes](#).

The [fix_modify virial](#) option is supported by this fix to add the contribution due to the pitorsion interactions to both the global pressure and per-atom stress of the system via the [compute pressure](#) and [compute stress/atom](#) commands. The former can be accessed by [thermodynamic output](#). The default setting for this fix is [fix_modify virial yes](#).

This fix computes a global scalar which can be accessed by various [output commands](#). The scalar is the potential energy discussed above. The scalar value calculated by this fix is “extensive”.

No parameter of this fix can be used with the *start/stop* keywords of the [run](#) command.

The forces due to this fix are imposed during an energy minimization, invoked by the [minimize](#) command.

The [fix_modify respa](#) option is supported by this fix. This allows to set at which level of the *r-RESPA* integrator the fix is adding its forces. Default is the outermost level.

Note

For energy minimization, if you want the potential energy associated with the pitorsion terms forces to be included in the total potential energy of the system (the quantity being minimized), you MUST not disable the [fix_modify energy](#) option for this fix.

2.10.5 Restrictions

To function as expected this fix command must be issued *before* a [read_data](#) command but *after* a [read_restart](#) command.

This fix can only be used if LAMMPS was built with the AMOEBA package. See the [Build package](#) page for more info.

2.10.6 Related commands

[fix_modify](#), [read_data](#)

2.10.7 Default

none

2.11 fix append/atoms command

2.11.1 Syntax

`fix ID group-ID append/atoms face ... keyword value ...`

- ID, group-ID are documented in [fix](#) command
- append/atoms = style name of this fix command
- face = *zhi*
- zero or more keyword/value pairs may be appended
- keyword = *basis* or *size* or *freq* or *temp* or *random* or *units*

basis values = M itype

M = which basis atom

itype = atom type (1-N) to assign to this basis atom

size args = Lz

Lz = z size of lattice region appended in a single event(distance units)

freq args = freq

freq = the number of timesteps between append events

temp args = target damp seed extent
 target = target temperature for the region between zhi-extent and zhi (temperature units)
 damp = damping parameter (time units)
 seed = random number seed for langevin kicks
 extent = extent of thermostatted region (distance units)
 random args = xmax ymax zmax seed
 xmax, ymax, zmax = maximum displacement in particular direction (distance units)
 seed = random number seed for random displacement
 units value = lattice or box
 lattice = the wall position is defined in lattice units
 box = the wall position is defined in simulation box units

2.11.2 Examples

```

fix 1 all append/atoms zhi size 5.0 freq 295 units lattice
fix 4 all append/atoms zhi size 15.0 freq 5 units box
fix A all append/atoms zhi size 1.0 freq 1000 units lattice

```

2.11.3 Description

This fix creates atoms on a lattice, appended on the zhi edge of the system box. This can be useful when a shock or wave is propagating from zlo. This allows the system to grow with time to accommodate an expanding wave. 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.

This fix will automatically freeze atoms on the zhi edge of the system, so that overlaps are avoided when new atoms are appended.

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 type = 1 unless this keyword specifies differently.

The *size* keyword defines the size in *z* of the chunk of material to be added.

The *random* keyword will give the atoms random displacements around their lattice points to simulate some initial temperature.

The *temp* keyword will cause a region to be thermostatted with a Langevin thermostat on the zhi boundary. The size of the region is measured from zhi and is set with the *extent* argument.

The *units* keyword determines the meaning of the distance units used to define a wall position, but only when a numeric constant is used. A *box* value selects standard distance units as defined by the [units](#) command (e.g., Å 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.

2.11.4 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](#).

2.11.5 Restrictions

This fix style is part of the SHOCK package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

The boundary on which atoms are added with append/atoms must be shrink/minimum. The opposite boundary may be any boundary type other than periodic.

2.11.6 Related commands

[fix wall/piston](#) command

2.11.7 Default

The keyword defaults are size = 0.0, freq = 0, units = lattice. All added atoms are of type 1 unless the basis keyword is used.

2.12 fix atc command

2.12.1 Syntax

```
fix <fixID> <group> atc <type> <parameter_file>
```

- fixID = name of fix
- group = name of group fix is to be applied
- type = *thermal* or *two_temperature* or *hardy* or *field*

thermal = thermal coupling with fields: temperature

two_temperature = electron-phonon coupling with field: temperature and electron _ temperature

hardy = on-the-fly post-processing using kernel localization functions

field = on-the-fly post-processing using mesh-based localization functions

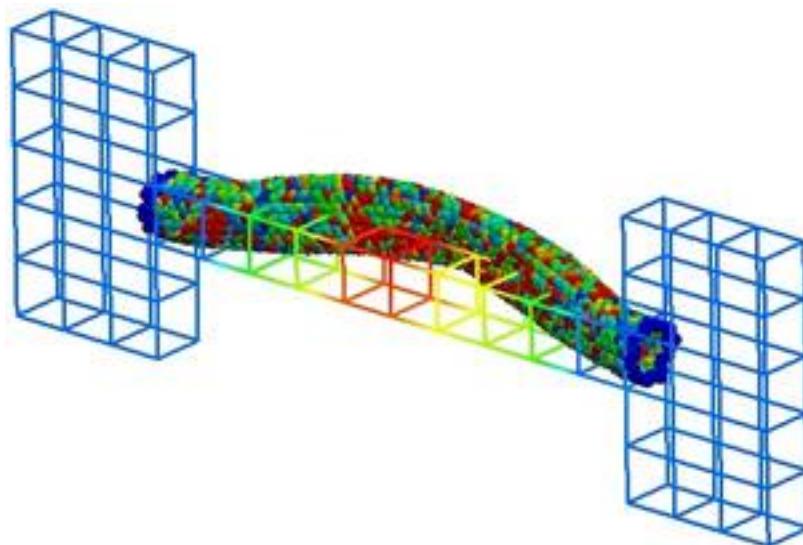
- parameter_file = name of the file with material parameters. Note: Neither hardy nor field requires a parameter file

2.12.2 Examples

```
fix AtC internal atc thermal Ar_thermal.dat
fix AtC internal atc two_temperature Ar_ttm.mat
fix AtC internal atc hardy
fix AtC internal atc field
```

2.12.3 Description

This fix is the beginning to creating a coupled FE/MD simulation and/or an on-the-fly estimation of continuum fields. The coupled versions of this fix do Verlet integration and the post-processing does not. After instantiating this fix, several other fix_modify commands will be needed to set up the problem (i.e., define the finite element mesh and prescribe initial and boundary conditions).



The following coupling example is typical, but non-exhaustive:

```
# ... commands to create and initialize the MD system

# initial fix to designate coupling type and group to apply it to
# tag group physics material_file
fix AtC internal atc thermal Ar_thermal.mat

# create a uniform 12 x 2 x 2 mesh that covers region contain the group
# nx ny nz region periodicity
fix_modify AtC mesh create 12 2 2 mdRegion f p p

# specify the control method for the type of coupling
# physics control_type
fix_modify AtC thermal control flux

# specify the initial values for the empirical field "temperature"
# field node_group value
fix_modify AtC initial temperature all 30
```

(continues on next page)

(continued from previous page)

```
# create an output stream for nodal fields
# filename output_frequency
fix_modify AtC output atc_fe_output 100

run 1000
```

likewise for this post-processing example:

```
# ... commands to create and initialize the MD system

# initial fix to designate post-processing and the group to apply it to
# no material file is allowed nor required
fix AtC internal atc hardy

# for hardy fix, specific kernel function (function type and range) to # be used as a localization function
fix AtC kernel quartic_sphere 10.0

# create a uniform 1 x 1 x 1 mesh that covers region contain the group
# with periodicity this effectively creates a system average
fix_modify AtC mesh create 1 1 1 box p p p

# change from default lagrangian map to eulerian
# refreshed every 100 steps
fix_modify AtC atom_element_map eulerian 100

# start with no field defined
# add mass density, potential energy density, stress and temperature
fix_modify AtC fields add density energy stress temperature

# create an output stream for nodal fields
# filename output_frequency
fix_modify AtC output nvtFE 100 text

run 1000
```

the mesh's linear interpolation functions can be used as the localization function by using the field option:

```
fix AtC internal atc field
fix_modify AtC mesh create 1 1 1 box p p p
...
```

Note coupling and post-processing can be combined in the same simulations using separate fixes.

2.12.4 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 not supported by this fix, but this fix does add the kinetic energy imparted to atoms by the momentum coupling mode of the AtC package to the global potential energy of the system as part of *thermodynamic output*.

Additional *fix_modify* options relevant to this fix are listed below.

This fix computes a global scalar which can be accessed by various *output commands*. The scalar is the energy discussed in the previous paragraph. The scalar value 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*.

2.12.5 Restrictions

Thermal and two_temperature (coupling) types use a Verlet time-integration algorithm. The hardy type does not contain its own time-integrator and must be used with a separate fix that does contain one (e.g., nve, nvt). In addition, currently:

- the coupling is restricted to thermal physics
- the FE computations are done in serial on each processor.

2.12.6 Related commands

After specifying this fix in your input script, several *fix_modify AtC* commands are used to setup the problem (e.g., define the finite element mesh and prescribe initial and boundary conditions). Each of these options has its own doc page.

fix_modify commands for setup:

- *fix_modify AtC mesh create*
- *fix_modify AtC mesh quadrature*
- *fix_modify AtC mesh read*
- *fix_modify AtC mesh write*
- *fix_modify AtC mesh create_nodeset*
- *fix_modify AtC mesh add_to_nodeset*
- *fix_modify AtC mesh create_faceset_box*
- *fix_modify AtC mesh create_faceset_plane*
- *fix_modify AtC mesh create_elementset*
- *fix_modify AtC mesh delete_elements*
- *fix_modify AtC nodeset_to_elementset*
- *fix_modify AtC boundary type*
- *fix_modify AtC internal_quadrature*
- *fix_modify AtC time_integration*
- *fix_modify AtC extrinsic electron_integration*
- *fix_modify AtC internal_element_set*

- *fix_modify AtC decomposition*

fix_modify commands for boundary and initial conditions:

- *fix_modify AtC initial*
- *fix_modify AtC fix*
- *fix_modify AtC unfix*
- *fix_modify AtC fix_flux*
- *fix_modify AtC unfix_flux*
- *fix_modify AtC source*
- *fix_modify AtC remove_source*

fix_modify commands for control and filtering:

- *fix_modify AtC control thermal*
- *fix_modify AtC control momentum*
- *fix_modify AtC control localized_lambda*
- *fix_modify AtC control lumped_lambda_solve*
- *fix_modify AtC control mask_direction*
- *fix_modify AtC filter*
- *fix_modify AtC filter scale*
- *fix_modify AtC filter type*
- *fix_modify AtC equilibrium_start*
- *fix_modify AtC extrinsic exchange*
- *fix_modify AtC poisson_solver*

fix_modify commands for output:

- *fix_modify AtC output*
- *fix_modify AtC output nodeset*
- *fix_modify AtC output volume_integral*
- *fix_modify AtC output boundary_integral*
- *fix_modify AtC output contour_integral*
- *fix_modify AtC mesh output*
- *fix_modify AtC write_restart*
- *fix_modify AtC read_restart*

fix_modify commands for post-processing:

- *fix_modify AtC kernel*
- *fix_modify AtC fields*
- *fix_modify AtC gradients*
- *fix_modify AtC rates*
- *fix_modify AtC computes*

- *fix_modify AtC on_the_fly*
- *fix_modify AtC pair/bond_interactions*
- *fix_modify AtC sample_frequency*
- *fix_modify AtC set*

miscellaneous *fix_modify* commands:

- *fix_modify AtC atom_element_map*
- *fix_modify AtC atom_weight*
- *fix_modify AtC write_atom_weights*
- *fix_modify AtC kernel_bandwidth*
- *fix_modify AtC reset_time*
- *fix_modify AtC reset_atomic_reference_positions*
- *fix_modify AtC fe_md_boundary*
- *fix_modify AtC boundary_faceset*
- *fix_modify AtC consistent_fe_initialization*
- *fix_modify AtC mass_matrix*
- *fix_modify AtC material*
- *fix_modify AtC atomic_charge*
- *fix_modify AtC source_integration*
- *fix_modify AtC temperature_definition*
- *fix_modify AtC track_displacement*
- *fix_modify AtC boundary_dynamics*
- *fix_modify AtC add_species*
- *fix_modify AtC add_molecule*
- *fix_modify AtC remove_species*
- *fix_modify AtC remove_molecule*

Note: a set of example input files with the attendant material files are included in the examples/PACKAGES/atc folders.

2.12.7 Default

None

For detailed exposition of the theory and algorithms please see:

(Wagner) Wagner, GJ; Jones, RE; Templeton, JA; Parks, MA, “An atomistic-to-continuum coupling method for heat transfer in solids.” Special Issue of Computer Methods and Applied Mechanics (2008) 197:3351.

(Zimmerman2004) Zimmerman, JA; Webb, EB; Hoyt, JJ.; Jones, RE; Klein, PA; Bammann, DJ, “Calculation of stress in atomistic simulation.” Special Issue of Modelling and Simulation in Materials Science and Engineering (2004), 12:S319.

(Zimmerman2010) Zimmerman, JA; Jones, RE; Templeton, JA, “A material frame approach for evaluating continuum variables in atomistic simulations.” Journal of Computational Physics (2010), 229:2364.

(Templeton2010) Templeton, JA; Jones, RE; Wagner, GJ, “Application of a field-based method to spatially varying thermal transport problems in molecular dynamics.” Modelling and Simulation in Materials Science and Engineering (2010), 18:085007.

(Jones) Jones, RE; Templeton, JA; Wagner, GJ; Olmsted, D; Modine, JA, “Electron transport enhanced molecular dynamics for metals and semi-metals.” International Journal for Numerical Methods in Engineering (2010), 83:940.

(Templeton2011) Templeton, JA; Jones, RE; Lee, JW; Zimmerman, JA; Wong, BM, “A long-range electric field solver for molecular dynamics based on atomistic-to-continuum modeling.” Journal of Chemical Theory and Computation (2011), 7:1736.

(Mandadapu) Mandadapu, KK; Templeton, JA; Lee, JW, “Polarization as a field variable from molecular dynamics simulations.” Journal of Chemical Physics (2013), 139:054115.

Please refer to the standard finite element (FE) texts (e.g., T.J.R. Hughes, *The Finite Element Method*, Dover 2003) for the basics of FE simulations.

2.13 fix atom/swap command

2.13.1 Syntax

```
fix ID group-ID atom/swap N X seed T keyword values ...
```

- ID, group-ID are documented in [fix](#) command
- atom/swap = style name of this fix command
- N = invoke this fix every N steps
- X = number of swaps to attempt every N steps
- seed = random # seed (positive integer)
- T = scaling temperature of the MC swaps (temperature units)
- one or more keyword/value pairs may be appended to args
- keyword = *types* or *mu* or *ke* or *semi-grand* or *region*

types values = two or more atom types (1-Ntypes or type label)

mu values = chemical potential of swap types (energy units)

ke value = no or yes

 no = no conservation of kinetic energy after atom swaps

 yes = kinetic energy is conserved after atom swaps

semi-grand value = no or yes

 no = particle type counts and fractions conserved

 yes = semi-grand canonical ensemble, particle fractions not conserved

region value = region-ID

 region-ID = ID of region to use as an exchange/move volume

2.13.2 Examples

```
fix 2 all atom/swap 1 1 29494 300.0 ke no types 1 2
fix myFix all atom/swap 100 1 12345 298.0 region my_swap_region types 5 6
fix SGMC all atom/swap 1 100 345 1.0 semi-grand yes types 1 2 3 mu 0.0 4.3 -5.0
```

2.13.3 Description

This fix performs Monte Carlo swaps of atoms of one given atom type with atoms of the other given atom types. The specified scaling temperature T is used in the Metropolis criterion dictating swap probabilities.

Perform X swaps of atoms of one type with atoms of another type according to a Monte Carlo probability. Swap candidates must be in the fix group, must be in the region (if specified), and must be of one of the listed types. Swaps are attempted between candidates that are chosen randomly with equal probability among the candidate atoms. Swaps are not attempted between atoms of the same type since nothing would happen.

All atoms in the simulation domain can be moved using regular time integration displacements (e.g., via [fix nvt](#)), resulting in a hybrid MC+MD simulation. A smaller-than-usual timestep size may be needed when running such a hybrid simulation, especially if the swapped atoms are not well equilibrated.

The *types* keyword is required. At least two atom types must be specified. If not using *semi-grand*, exactly two atom types are required.

The *ke* keyword can be set to *no* to turn off kinetic energy conservation for swaps. The default is *yes*, which means that swapped atoms have their velocities scaled by the ratio of the masses of the swapped atom types. This ensures that the kinetic energy of each atom is the same after the swap as it was before the swap, even though the atom masses have changed.

The *semi-grand* keyword can be set to *yes* to switch to the semi-grand canonical ensemble as discussed in ([Sadigh](#)). This means that the total number of each particle type does not need to be conserved. The default is *no*, which means that the only kind of swap allowed exchanges an atom of one type with an atom of a different given type. In other words, the relative mole fractions of the swapped atoms remains constant. Whereas in the semi-grand canonical ensemble, the composition of the system can change. Note that when using *semi-grand*, atoms in the fix group whose type is not listed in the *types* keyword are ineligible for attempted conversion. An attempt is made to switch the selected atom (if eligible) to one of the other listed types with equal probability. Acceptance of each attempt depends upon the Metropolis criterion.

The *mu* keyword allows users to specify chemical potentials. This is required and allowed only when using *semi-grand*. All chemical potentials are absolute, so there is one for each swap type listed following the *types* keyword. In semi-grand canonical ensemble simulations the chemical composition of the system is controlled by the difference in these values. So shifting all values by a constant amount will have no effect on the simulation.

This command may optionally use the *region* keyword to define swap volume. The specified region must have been previously defined with a [region](#) command. It must be defined with side = *in*. Swap attempts occur only between atoms that are both within the specified region. Swaps are not otherwise attempted.

You should ensure you do not swap atoms belonging to a molecule, or LAMMPS will eventually generate an error when it tries to find those atoms. LAMMPS will warn you if any of the atoms eligible for swapping have a non-zero molecule ID, but does not check for this at the time of swapping.

If not using *semi-grand* this fix checks to ensure all atoms of the given types have the same atomic charge. LAMMPS does not enforce this in general, but it is needed for this fix to simplify the swapping procedure. Successful swaps will swap the atom type and charge of the swapped atoms. Conversely, when using *semi-grand*, it is assumed that all the atom types involved in switches have the same charge. Otherwise, charge would not be conserved. As a consequence, no checks on atomic charges are performed, and successful switches update the atom type but not the atom charge. While it is possible to use *semi-grand* with groups of atoms that have different charges, these charges will not be changed when the atom types change.

Since this fix computes total potential energies before and after proposed swaps, so even complicated potential energy calculations are OK, including the following:

- long-range electrostatics (k -space)
- many body pair styles
- hybrid pair styles
- eam pair styles
- triclinic systems
- need to include potential energy contributions from other fixes

Some fixes have an associated potential energy. Examples of such fixes include: [efield](#), [gravity](#), [addforce](#), [langevin](#), [restrain](#), [temp/berendsen](#), [temp/rescale](#), and [wall](#) fixes. For that energy to be included in the total potential energy of the system (the quantity used when performing GCMC moves), you **must** enable the [fix_modify](#) energy option for that fix. The doc pages for individual [fix](#) commands specify if this should be done.

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

This fix writes the state of the fix to [binary restart files](#). This includes information about the random number generator seed, the next timestep for MC exchanges, the number of exchange attempts and successes, etc. See the [read_restart](#) command for info on how to re-specify a fix in an input script that reads a restart file, so that the operation of the fix continues in an uninterrupted fashion.

Note

For this to work correctly, the timestep must **not** be changed after reading the restart with [reset_timestep](#). The fix will try to detect it and stop with an error.

None of the [fix_modify](#) options are relevant to this fix.

This fix computes a global vector of length 2, which can be accessed by various [output commands](#). The vector values are the following global cumulative quantities:

1. swap attempts
2. swap accepts

The vector values calculated by this fix are “intensive”.

No parameter of this fix can be used with the [start/stop](#) keywords of the [run](#) command. This fix is not invoked during [energy minimization](#).

2.13.5 Restrictions

This fix is part of the MC package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) doc page for more info.

This fix cannot be used with systems that do not have per-type masses (e.g. atom style sphere) since the implemented algorithm pre-computes velocity rescaling factors from per-type masses and ignores any per-atom masses, if present. In case both, per-type and per-atom masses are present, a warning is printed.

2.13.6 Related commands

fix nvt, neighbor, fix deposit, fix evaporate, delete_atoms, fix gcmc, fix mol/swap, fix sgcmc

2.13.7 Default

The option defaults are *ke* = yes, *semi-grand* = no, *mu* = 0.0 for all atom types.

(Sadigh) B Sadigh, P Erhart, A Stukowski, A Caro, E Martinez, and L Zepeda-Ruiz, Phys. Rev. B, 85, 184203 (2012).

2.14 fix ave/atom command

2.14.1 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] = *I*th column of per-atom array calculated by a compute with ID, *I* can include wildcard *_* (see below)
f_ID = per-atom vector calculated by a fix with ID
f_ID[I] = *I*th column of per-atom array calculated by a fix with ID, *I* can include wildcard (see *_* below)
v_name = per-atom vector calculated by an atom-style variable with name

2.14.2 Examples

```
fix 1 all ave/atom 1 100 100 vx vy vz
fix 1 all ave/atom 10 20 1000 c_my_stress[1]
fix 1 all ave/atom 10 20 1000 c_my_stress[*]
```

2.14.3 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/chunk](#) 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.

Each per-atom value of each input vector is averaged independently.

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.

Note that for values from a compute or fix, the bracketed index I can be specified using a wildcard asterisk with the index to effectively specify multiple values. This takes the form “*” or “*n” or “m*” or “m*n”. If N is the size of the vector (for *mode* = scalar) or the number of columns in the array (for *mode* = vector), then an asterisk with no numeric values means all indices from 1 to N . A leading asterisk means all indices from 1 to n (inclusive). A trailing asterisk means all indices from m to N (inclusive). A middle asterisk means all indices from m to n (inclusive).

Using a wildcard is the same as if the individual columns of the array had been listed one by one. For example, these two fix ave/atom commands are equivalent, since the [compute stress/atom](#) command creates a per-atom array with six columns:

```
compute my_stress all stress/atom NULL
fix 1 all ave/atom 10 20 1000 c_my_stress[*]
fix 1 all ave/atom 10 20 1000 c_my_stress[1] c_my_stress[2] &
                  c_my_stress[3] c_my_stress[4] &
                  c_my_stress[5] c_my_stress[6]
```

The N_{every} , N_{repeat} , and N_{freq} 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 N_{freq} . The average is over N_{repeat} quantities, computed in the preceding portion of the simulation every N_{every} timesteps. N_{freq} must be a multiple of N_{every} and N_{every} must be non-zero even if N_{repeat} is 1. Also, the timesteps contributing to the average value cannot overlap; that is, $N_{\text{repeat}} \times N_{\text{every}}$ cannot exceed N_{freq} .

For example, if $N_{\text{every}} = 2$, $N_{\text{repeat}} = 6$, and $N_{\text{freq}} = 100$, then values on timesteps 90, 92, 94, 96, 98, and 100 will be used to compute the final average on time step 100. Similarly for timesteps 190, 192, 194, 196, 198, and 200 on time step 200, etc.

The atom attribute values (x , y , z , vx , vy , vz , fx , fy , and 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.

Note

The x , y , and 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 of 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*, and *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 *I*th column of the per-atom array calculated by the compute is used. Users can also write code for their own compute styles and [add them to LAMMPS](#). See the discussion above for how *I* can be specified with a wildcard asterisk to effectively specify multiple values.

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 *I*th column of the per-atom array calculated by the fix is used. Note that some fixes only produce their values on certain timesteps, which must be compatible with *Nevery*, else an error will result. Users can also write code for their own fix styles and [add them to LAMMPS](#). See the discussion above for how *I* can be specified with a wildcard asterisk to effectively specify multiple values.

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.

2.14.4 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](#).

2.14.5 Restrictions

none

2.14.6 Related commands

[compute](#), [fix ave/histo](#), [fix ave/chunk](#), [fix ave/time](#), [variable](#),

2.14.7 Default

none

2.15 fix ave/chunk command

2.15.1 Syntax

```
fix ID group-ID ave/chunk Nevery Nrepeat Nfreq chunkID value1 value2 ... keyword args ...
```

- ID, group-ID are documented in [fix](#) command
- ave/chunk = 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
- chunkID = ID of [compute chunk/atom](#) command
- one or more input values can be listed
- value = vx, vy, vz, fx, fy, fz, density/mass, density/number, mass, temp, c_ID, c_ID[I], f_ID, f_ID[I], v_name

vx,vy,vz,fx,fy,fz,mass = atom attribute (velocity, force component, mass)
 density/number, density/mass = number or mass density (per volume)
 temp = temperature
 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, I can include wildcard [\(see below\)](#)
 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, I can include wildcard [\(see below\)](#)
 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 *ave* or *bias* or *adof* or *c dof* or *file* or *append* or *overwrite* or *format* or *title1* or *title2* or *title3*
- norm arg = all or sample or none = how output on Nfreq steps is normalized
 all = output is sum of atoms across all Nrepeat samples, divided by atom count
 sample = output is sum of Nrepeat sample averages, divided by Nrepeat
 none = output is sum of Nrepeat sample sums, divided by Nrepeat
 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
 bias arg = bias-ID
 bias-ID = ID of a temperature compute that removes a velocity bias for temperature calculation
 adof value = dof_per_atom
 dof_per_atom = define this many degrees-of-freedom per atom for temperature calculation
 cdof value = dof_per_chunk
 dof_per_chunk = define this many degrees-of-freedom per chunk for temperature calculation
 file arg = filename

```

filename = file to write results to
append arg = filename
filename = file to append results to
overwrite arg = none = overwrite output file with only latest output
format arg = string
string = C-style format string
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

```

2.15.2 Examples

```

fix 1 all ave/chunk 10000 1 10000 binchunk c_myCentro title1 "My output values"
fix 1 flow ave/chunk 100 10 1000 molchunk vx vz norm sample file vel.profile
fix 1 flow ave/chunk 100 5 1000 binchunk density/mass ave running
fix 1 flow ave/chunk 100 5 1000 binchunk density/mass ave running

```

Note

Changed in version 31May2016.

If you are trying to replace a deprecated fix ave/spatial command with the newer, more flexible fix ave/chunk and [compute chunk/atom](#) commands, you simply need to split the fix ave/spatial arguments across the two new commands. For example, this command:

```
fix 1 flow ave/spatial 100 10 1000 y 0.0 1.0 vx vz norm sample file vel.profile
```

could be replaced by:

```
compute cc1 flow chunk/atom bin/1d y 0.0 1.0
fix 1 flow ave/chunk 100 10 1000 cc1 vx vz norm sample file vel.profile
```

2.15.3 Description

Use one or more per-atom vectors as inputs every few timesteps, sum the values over the atoms in each chunk at each timestep, then average the per-chunk values over longer timescales. The resulting chunk averages can be used by other [output commands](#) such as [thermo_style custom](#), and can also be written to a file.

In LAMMPS, chunks are collections of atoms defined by a [compute chunk/atom](#) command, which assigns each atom to a single chunk (or no chunk). The ID for this command is specified as chunkID. For example, a single chunk could be the atoms in a molecule or atoms in a spatial bin. See the [compute chunk/atom](#) page and the [Howto chunk](#) page for details of how chunks can be defined and examples of how they can be used to measure properties of a system.

Note that if the [compute chunk/atom](#) command defines spatial bins, the fix ave/chunk command performs a similar computation as the [fix ave/grid](#) command. However, the per-bin outputs from the fix ave/chunk command are global; each processor stores a copy of the entire set of bin data. By contrast, the [fix ave/grid](#) command uses a distributed grid where each processor owns a subset of the bins. Thus it is more efficient to use the [fix ave/grid](#) command when the grid is large and a simulation is run on many processors.

Note that only atoms in the specified group contribute to the summing and averaging calculations. The [compute chunk/atom](#) command defines its own group as well as an optional region. Atoms will have a chunk ID = 0, meaning they belong to no chunk, if they are not in that group or region. Thus you can specify the “all” group for this command if you simply want to use the chunk definitions provided by chunkID.

Each specified per-atom value can be an atom attribute (position, velocity, force component), a number or mass density, a mass or temperature, 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. Note that the [compute property/atom](#) command provides access to any attribute defined and stored by atoms. If you wish to time-average global quantities from a compute, fix, or variable, then see the [fix ave/time](#) command.

The per-atom values of each input vector are summed and averaged independently of the per-atom values in other input vectors.

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.

Note that for values from a compute or fix that produces a per-atom array (multiple values per atom), the bracketed index I can be specified using a wildcard asterisk with the index to effectively specify multiple values. This takes the form “*” or “*n” or “n*” or “m*n”. If N = the size of the vector (for mode = scalar) or the number of columns in the array (for mode = vector), then an asterisk with no numeric values means all indices from 1 to N . A leading asterisk means all indices from 1 to n (inclusive). A trailing asterisk means all indices from m to N (inclusive). A middle asterisk means all indices from m to n (inclusive).

Using a wildcard is the same as if the individual columns of the array had been listed one by one. For example, these two fix ave/chunk commands are equivalent, since the [compute property/atom](#) command creates, in this case, a per-atom array with three columns:

```
compute myAng all property/atom angmomx angmomy angmomz
fix 1 all ave/chunk 100 1 100 cc1 c_myAng[*] file tmp.angmom
fix 2 all ave/chunk 100 1 100 cc1 c_myAng[1] c_myAng[2] c_myAng[3] file tmp.angmom
```

Note

This fix works by creating an array of size $N_{\text{chunk}} \times N_{\text{values}}$ on each processor. N_{chunk} is the number of chunks, which is defined by the [compute chunk/atom](#) command. N_{values} is the number of input values specified. Each processor loops over its atoms, tallying its values to the appropriate chunk. Then the entire array is summed across all processors. This means that using a large number of chunks will incur an overhead in memory and computational cost (summing across processors), so be careful to define a reasonable number of chunks.

The N_{every} , N_{repeat} , and N_{freq} arguments specify on what time steps the input values will be accessed and contribute to the average. The final averaged quantities are generated on time steps that are multiples of N_{freq} . The average is over N_{repeat} quantities, computed in the preceding portion of the simulation every N_{every} time steps. N_{freq} must be a multiple of N_{every} and N_{every} must be non-zero even if $N_{\text{repeat}} = 1$. Also, the time steps contributing to the average value cannot overlap (i.e., $N_{\text{repeat}} \times N_{\text{every}}$ cannot exceed N_{freq}).

For example, if $N_{\text{every}} = 2$, $N_{\text{repeat}} = 6$, and $N_{\text{freq}} = 100$, then values on time steps 90, 92, 94, 96, 98, 100 will be used to compute the final average on time step 100. Similarly for time steps 190, 192, 194, 196, 198, 200 on time step 200, etc. If $N_{\text{repeat}} = 1$ and $N_{\text{freq}} = 100$, then no time averaging is done; values are simply generated on time steps 100, 200, etc.

Each input value can also be averaged over the atoms in each chunk. The way the averaging is done across the N_{repeat} time steps to produce output on the N_{freq} time steps, and across multiple N_{freq} outputs, is determined by the *norm* and *ave* keyword settings, as discussed below.

Note

To perform per-chunk averaging within a N_{freq} time window, the number of chunks N_{chunk} defined by the [compute chunk/atom](#) command must remain constant. If the *ave* keyword is set to *running* or *window* then N_{chunk} must remain constant for the duration of the simulation. This fix forces the chunk/atom compute specified by chunkID to hold N_{chunk} constant for the appropriate time windows, by not allowing it to re-calculate N_{chunk} , which can also affect how it assigns chunk IDs to atoms. This is particularly important to understand if the chunks defined by the [compute chunk/atom](#) command are spatial bins. If its *units* keyword is set to *box* or *lattice*, then the number of bins N_{chunk} and size of each bin will be fixed over the N_{freq} time window, which can affect which atoms are discarded if the simulation box size changes. If its *units* keyword is set to *reduced*, then the number of bins N_{chunk} will still be fixed, but the size of each bin can vary at each time step if the simulation box size changes (e.g., for an NPT simulation).

The atom attribute values (*vx*, *vy*, *vz*, *fx*, , *fz*, *mass*) are self-explanatory. As noted above, any other atom attributes can be used as input values 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 for each chunk (i.e., number/volume). The *density/mass* value means the mass density is computed for each chunk (i.e., total-mass/volume). The output values are in units of 1/volume or mass density (mass/volume). See the [units](#) command page for the definition of density for each choice of units (e.g., g/cm³). If the chunks defined by the [compute chunk/atom](#) command are spatial bins, the volume is the bin volume. Otherwise, it is the volume of the entire simulation box.

The *temp* value means the temperature is computed for each chunk, by the formula

$$\text{KE} = \frac{\text{DOF}}{2} k_B T,$$

where KE is the total kinetic energy of the chunk of atoms (sum of $\frac{1}{2}mv^2$), DOF is the the total number of degrees of freedom for all atoms in the chunk, k_B is the Boltzmann constant, and T is the absolute temperature.

The DOF is calculated as $N * \text{adof} + \text{cdof}$, where N is the number of atoms in the chunk, adof is the number of degrees of freedom per atom, and cdof is the number of degrees of freedom per chunk. By default, adof = 2 or 3 = dimensionality of system, as set via the [dimension](#) command, and cdof = 0.0. This gives the usual formula for temperature.

Note that currently this temperature only includes translational degrees of freedom for each atom. No rotational degrees of freedom are included for finite-size particles. Also, no degrees of freedom are subtracted for any velocity bias or constraints that are applied, such as [compute temp/partial](#), [fix shake](#), or [fix rigid](#). This is because those degrees of freedom (e.g., a constrained bond) could apply to sets of atoms that are both included and excluded from a specific chunk, and hence the concept is somewhat ill-defined. In some cases, you can use the *adof* and *cdof* keywords to adjust the calculated degrees of freedom appropriately, as explained below.

Also note that a bias can be subtracted from atom velocities before they are used in the above formula for KE, by using the *bias* keyword. This allows, for example, a thermal temperature to be computed after removal of a flow velocity profile.

Note that the per-chunk temperature calculated by this fix and the [compute temp/chunk](#) command can be different. The compute calculates the temperature for each chunk for a single snapshot. This fix can do that but can also time average those values over many snapshots, or it can compute a temperature as if the atoms in the chunk on different time steps were collected together as one set of atoms to calculate their temperature. The compute allows the center-of-mass velocity of each chunk to be subtracted before calculating the temperature; this fix does not.

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 LAMMPS](#). See the discussion above for how I can be specified with a wildcard asterisk to effectively specify multiple values.

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 time steps, which must be compatible with N_{every} , else an error results. Users can also write code for their own fix styles and [add them to LAMMPS](#). See the discussion above for how I can be specified with a wildcard asterisk to effectively specify multiple values.

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 average within chunks.

Additional optional keywords also affect the operation of this fix and its outputs.

The *norm* keyword affects how averaging is done for the per-chunk values that are output every N_{freq} time steps.

If the *norm* setting is *all*, which is the default, a chunk value is summed over all atoms in all N_{repeat} samples, as is the count of atoms in the chunk. The averaged output value for the chunk on the N_{freq} time steps is Total-sum / Total-count. In other words it is an average over atoms across the entire N_{freq} timescale. For the *density/number* and *density/mass* values, the volume (bin volume or system volume) used in the final normalization will be the volume at the final N_{freq} time step. For the *temp* values, degrees of freedom and kinetic energy are summed separately across the entire N_{freq} timescale, and the output value is calculated by dividing those two sums.

If the *norm* setting is *sample*, the chunk value is summed over atoms for each sample, as is the count, and an “average sample value” is computed for each sample (i.e., Sample-sum / Sample-count). The output value for the chunk on the N_{freq} time steps is the average of the N_{repeat} “average sample values” (i.e., the sum of N_{repeat} “average sample values” divided by N_{repeat}). In other words, it is an average of an average. For the *density/number* and *density/mass* values, the volume (bin volume or system volume) used in the per-sample normalization will be the current volume at each sampling step.

If the *norm* setting is *none*, a similar computation as for the *sample* setting is done, except the individual “average sample values” are “summed sample values”. A summed sample value is simply the chunk value summed over atoms in the sample, without dividing by the number of atoms in the sample. The output value for the chunk on the N_{freq} timesteps is the average of the N_{repeat} “summed sample values” (i.e., the sum of N_{repeat} “summed sample values” divided by N_{repeat}). For the *density/number* and *density/mass* values, the volume (bin volume or system volume) used in the per-sample sum normalization will be the current volume at each sampling step.

The *ave* keyword determines how the per-chunk values produced every N_{freq} steps are averaged with values produced on previous steps that were multiples of N_{freq} , before they are accessed by another output command or written to a file.

If the *ave* setting is *one*, which is the default, then the chunk values produced on timesteps that are multiples of N_{freq} are independent of each other; they are output as-is without further averaging.

If the *ave* setting is *running*, then the chunk values produced on timesteps that are multiples of N_{freq} are summed and averaged in a cumulative sense before being output. Each output chunk value is thus the average of the chunk value produced on that timestep with all preceding values for the same chunk. 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 chunk values produced on timesteps that are multiples of N_{freq} are summed and averaged within a moving “window” of time, so that the last M values for the same chunk are used to produce the output. For example, if $M = 3$ and $N_{\text{freq}} = 1000$, then the output on step 10000 will be the average of the individual chunk values on time steps 8000, 9000, and 10000. Outputs on early steps will average over less than M values if they are not available.

The *bias* keyword specifies the ID of a temperature compute that removes a “bias” velocity from each atom, specified as *bias-ID*. It is only used when the *temp* value is calculated, to compute the thermal temperature of each chunk after the translational kinetic energy components have been altered in a prescribed way (e.g., to remove a flow velocity profile). See the doc pages for individual computes that calculate a temperature to see which ones implement a bias.

The *adof* and *c dof* keywords define the values used in the degree of freedom (DOF) formula described above for temperature calculation for each chunk. They are only used when the *temp* value is calculated. They can be used to calculate a more appropriate temperature for some kinds of chunks. Here are three examples:

If spatially binned chunks contain some number of water molecules and *fix shake* is used to make each molecule rigid, then you could calculate a temperature with six degrees of freedom (DOF) (three translational, three rotational) per molecule by setting *adof* to 2.0.

If *compute temp/partial* is used with the *bias* keyword to only allow the *x* component of velocity to contribute to the temperature, then *adof* = 1.0 would be appropriate.

If each chunk consists of a large molecule, with some number of its bonds constrained by *fix shake* or the entire molecule by *fix rigid/small*, *adof* = 0.0 and *c dof* could be set to the remaining degrees of freedom for the entire molecule (entire chunk in this case), that is, 6 for 3d or 3 for 2d for a rigid molecule.

Added in version 17Apr2024: new keyword *append*

The *file* or *append* keywords allow a filename to be specified. 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. Every N_{freq} timesteps, a section of chunk info will be written to a text file in the following format. A line with the timestep and number of chunks is written. Then one line per chunk is written, containing the chunk ID ($1 - N_{\text{chunk}}$), an optional original ID value, optional coordinate values for chunks that represent spatial bins, the number of atoms in the chunk, and one or more calculated values. More explanation of the optional values is given below. The number of values in each line corresponds to the number of values specified in the fix ave/chunk command. The number of atoms and the value(s) are summed or average quantities, as explained above.

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 *format* keyword sets the numeric format of each value when it is printed to a file via the *file* keyword. Note that all values are floating point quantities. The default format is `%g`. You can specify a higher precision if desired (e.g., `%.20.16g`).

The *title1* and *title2* and *title3* keywords allow specification of the strings that will be printed as the first three lines of the output file, assuming the *file* keyword was used. LAMMPS uses default values for each of these, so they do not need to be specified.

By default, these header lines are as follows:

```
# Chunk-averaged data for fix ID and group name
# Timestep Number-of-chunks
# Chunk (OrigID) (Coord1) (Coord2) (Coord3) Ncount 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 value names (e.g., *fx* or *c_myCompute[2]*).

The words in parenthesis only appear with corresponding columns if the chunk style specified for the *compute chunk/atom* command supports them. The OrigID column is only used if the *compress* keyword was set to yes for

the *compute chunk/atom* command. This means that the original chunk IDs (e.g., molecule IDs) will have been compressed to remove chunk IDs with no atoms assigned to them. Thus a compressed chunk ID of 3 may correspond to an original chunk ID or molecule ID of 415. The OrigID column will list 415 for the third chunk.

The CoordN columns only appear if a *binning* style was used in the *compute chunk/atom* command. For *bin/1d*, *bin/2d*, and *bin/3d* styles the column values are the center point of the bin in the corresponding dimension. Just Coord1 is used for *bin/1d*, Coord2 is added for *bin/2d*, Coord3 is added for *bin/3d*. For *bin/sphere*, just Coord1 is used, and it is the radial coordinate. For *bin/cylinder*, Coord1 and Coord2 are used. Coord1 is the radial coordinate (away from the cylinder axis), and coord2 is the coordinate along the cylinder axis.

Note that if the value of the *units* keyword used in the *compute chunk/atom command* is *box* or *lattice*, the coordinate values will be in distance *units*. If the value of the *units* keyword is *reduced*, the coordinate values will be in unitless reduced units (0–1). This is not true for the Coord1 value of style *bin/sphere* or *bin/cylinder* which both represent radial dimensions. Those values are always in distance *units*.

2.15.4 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 N_{freq} , since that is when averaging is performed. The global array has # of rows = the number of chunks N_{chunk} , as calculated by the specified *compute chunk/atom* command. The # of columns is $M + 1 + N_{\text{values}}$, where $M \in \{1, \dots, 4\}$, depending on whether the optional columns for OrigID and CoordN are used, as explained above. Following the optional columns, the next column contains the count of atoms in the chunk, and the remaining columns are the Nvalue quantities. When the array is accessed with a row I that exceeds the current number of chunks, than a 0.0 is returned by the fix instead of an error, since the number of chunks can vary as a simulation runs depending on how that value is computed by the *compute chunk/atom* command.

The array values calculated by this fix are treated as “intensive”, since they are typically already normalized by the count of atoms in each chunk.

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*.

2.15.5 Restrictions

none

2.15.6 Related commands

compute, *fix ave/atom*, *fix ave/histo*, *fix ave/time*, *variable*, *fix ave/correlate*, *fix ave/grid*

2.15.7 Default

The option defaults are norm = all, ave = one, bias = none, no file output, and title 1,2,3 = strings as described above.

2.16 fix ave/correlate command

2.16.1 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 time 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, I can include wildcard [_](#)
(see below)
f_ID = global scalar calculated by a fix with ID
f_ID[I] = Ith component of global vector calculated by a fix with ID, I can include wildcard [_](#)
(see below)
v_name = global value calculated by an equal-style variable with name
v_name[I] = Ith component of a vector-style variable with name, I can include wildcard (see below)

- 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 + lower
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

2.16.2 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"
fix 1 all ave/correlate 1 50 10000 c_thermo_press[*]
```

2.16.3 Description

Use one or more global scalar values as inputs every few timesteps, calculate time correlations between 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. See the *fix ave/correlate/long* command for an alternate method for computing correlation functions efficiently over very long time windows.

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 or vector-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/chunk*, *fix ave/atom*, or *fix ave/histo* commands. If you wish to convert a per-atom quantity into a single global value, see the *compute reduce* command.

The input values must be all scalars. What kinds of correlations between input values are calculated is determined by the *type* keyword as discussed below.

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* and *vector* are the only ones that can be used with this fix. Variables of style *atom* cannot be used, since they produce per-atom values.

For input values from a compute or fix or variable , the bracketed index I can be specified using a wildcard asterisk with the index to effectively specify multiple values. This takes the form “*” or “*n” or “m*” or “m*n”. If N is the size of the vector, then an asterisk with no numeric values means all indices from 1 to N. A leading asterisk means all indices from 1 to n (inclusive). A trailing asterisk means all indices from m to N (inclusive). A middle asterisk means all indices from m to n (inclusive).

Using a wildcard is the same as if the individual elements of the vector had been listed one by one. For example, the following two fix ave/correlate commands are equivalent, since the *compute pressure* command creates a global vector with six values:

```
compute myPress all pressure NULL
fix 1 all ave/correlate 1 50 10000 c_myPress[*]
fix 1 all ave/correlate 1 50 10000 &
    c_myPress[1] c_myPress[2] c_myPress[3] &
    c_myPress[4] c_myPress[5] c_myPress[6]
```

Note

For a vector-style variable, only the wildcard forms “*n” or “m*n” are allowed. You must specify the upper bound, because vector-style variable lengths are not determined until the variable is evaluated. If n is specified larger than the vector length turns out to be, zeroes are output for missing vector values.

The N_{every} , N_{repeat} , and N_{freq} arguments specify on what timesteps the input values will be used to calculate correlation data. The input values are sampled every N_{every} time steps. The correlation data for the preceding samples is computed on time steps that are a multiple of N_{freq} . 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 C_{ij} is calculated as:

$$C_{ij}(\Delta t) = \langle V_i(t)V_j(t + \Delta t) \rangle ,$$

which is the correlation value between input values V_i and V_j , separated by time Δt . Note that the second value V_j in the pair is always the one sampled at the later time. The average is an average over every pair of samples in the set that are separated by time Δt . The maximum Δt used is of size $(N_{\text{repeat}} - 1)N_{\text{every}}$. Thus the correlation between a pair of input values yields N_{repeat} correlation data:

$$C_{ij}(0), C_{ij}(N_{\text{every}}), C_{ij}(2N_{\text{every}}), \dots, C_{ij}((N_{\text{repeat}} - 1)N_{\text{every}})$$

For example, if $N_{\text{every}} = 5$, $N_{\text{repeat}} = 6$, and $N_{\text{freq}} = 100$, then values on time steps 0, 5, 10, 15, …, 100 will be used to compute the final averages on time step 100. Six averages will be computed: $C_{ij}(0)$, $C_{ij}(5)$, $C_{ij}(10)$, $C_{ij}(15)$, $C_{ij}(20)$, and $C_{ij}(25)$. $C_{ij}(10)$ on time step 100 will be the average of 19 samples, namely $V_i(0)V_j(10)$, $V_i(5)V_j(15)$, $V_i(10)V_j(20)$, $V_i(15)V_j(25)$, …, $V_i(85)V_j(95)$, and $V_i(90)V_j(100)$.

N_{freq} must be a multiple of N_{every} ; N_{every} and N_{repeat} must be non-zero. Also, if the *ave* keyword is set to *one* which is the default, then $N_{\text{freq}} \geq (N_{\text{repeat}} - 1)N_{\text{every}}$ 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 I^{th} element of the global vector calculated by the compute is used. See the discussion above for how I can be specified with a wildcard asterisk to effectively specify multiple values.

Note that there is a *compute reduce* command that can sum per-atom quantities into a global scalar or vector which can then be accessed by fix ave/correlate. It can also be a compute defined not in your input script, but by *thermodynamic output* or other fixes such as *fix nvt* or *fix temp/rescale*. See the doc pages for these commands which give the IDs of these computes. Users can also write code for their own compute styles and *add them to LAMMPS*.

If a value begins with “f_”, a fix ID must follow which has been previously defined in the input script. If no bracketed term is appended, the global scalar calculated by the fix is used. If a bracketed term is appended, the I^{th} element of the global vector calculated by the fix is used. See the discussion above for how I can be specified with a wildcard asterisk to effectively specify multiple values.

Note that some fixes only produce their values on certain timesteps, which must be compatible with N_{every} , else an error will result. Users can also write code for their own fix styles and *add them to LAMMPS*.

If a value begins with “v_”, a variable name must follow which has been previously defined in the input script. Only equal-style or vector-style variables can be referenced; the latter requires a bracketed term to specify the I^{th} element of the vector calculated by the variable. See the *variable* command for details. Note that variables of style *equal* or *vector* 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 V_i , with $i \in \{1, \dots, N\}$, let the number of pairs be N_{pair} . Note that the second value in the pair, $V_i(t)V_j(t + \Delta t)$, 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., $C_{ii} = V_i^2$ for $i \in \{1, \dots, N\}$, so $N_{\text{pair}} = N$).
- If *type* is set to *upper* then each input value is correlated with every succeeding value (i.e., $C_{ij} = V_iV_j$ for $i < j$, so $N_{\text{pair}} = N(N - 1)/2$).
- If *type* is set to *lower* then each input value is correlated with every preceding value (i.e., $C_{ij} = V_iV_j$ for $i > j$, so $N_{\text{pair}} = N(N - 1)/2$).
- If *type* is set to *auto/upper* then each input value is correlated with itself and every succeeding value (i.e., $C_{ij} = V_iV_j$ for $i \geq j$, so $N_{\text{pair}} = N(N + 1)/2$).
- If *type* is set to *auto/lower* then each input value is correlated with itself and every preceding value (i.e., $C_{ij} = V_iV_j$ for $i \leq j$, so $N_{\text{pair}} = N(N + 1)/2$).
- If *type* is set to *full* then each input value is correlated with itself and every other value (i.e., $C_{ij} = V_iV_j$ for $\{i, j\} = \{1, N\}$, so $N_{\text{pair}} = N^2$).

The *ave* keyword determines what happens to the accumulation of correlation samples every N_{freq} timesteps. If the *ave* setting is *one*, then the accumulation is restarted or zeroed every N_{freq} timesteps. Thus the outputs on successive N_{freq} timesteps are essentially independent of each other. The exception is that the $C_{ij}(0) = V_i(t)V_j(t)$ value at a time step t , where t is a multiple of N_{freq} , contributes to the correlation output both at time t and at time $t + N_{\text{freq}}$.

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 N_{every} 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 time step 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 V_iV_j , 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 N_{freq} steps, an array of correlation data is written to the file. The number of rows is N_{repeat} , as described above. The number of columns is $N_{\text{pair}} + 2$, 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*, *title2*, and *title3* keywords allow specification of the strings that will be printed as the first three lines of the output file, assuming the *file* keyword was used. LAMMPS uses default values for each of these, so they do not need to be specified.

By default, these header lines are as follows:

```
# Time-correlated data for fix ID
# TimeStep Number-of-time-windows
# Index TimeDelta Ncount valueI*valueJ valueI*valueJ ...
```

In the first line, ID is replaced with the fix-ID. The second line describes the two values that are printed at the first of each section of output. In the third line the value pairs are replaced with the appropriate fields from the fix ave/correlate command.

Let S_{ij} be a set of time correlation data for input values I and J , namely the N_{repeat} values:

$$S_{ij} = C_{ij}(0), C_{ij}(N_{\text{every}}), C_{ij}(2N_{\text{every}}), \dots, C_{ij}(N_{\text{repeat}} - 1)N_{\text{every}})$$

As explained below, these data 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 data, 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.

2.16.4 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 N_{freq} since that is when averaging is performed. The global array has # of rows N_{repeat} and # of columns $N_{\text{pair}} + 2$. The first column has the time Δt (in time steps) between the pairs of input values used to calculate the correlation, as described above. The second column has the number of samples contributing to the correlation average, as described above. The remaining N_{pair} columns are for I, J pairs of the N input values, as determined by the *type* keyword, as described above.

- For *type = auto*, the $N_{\text{pair}} = N$ columns are ordered: $C_{11}, C_{22}, \dots, C_{NN}$
- For *type = upper*, the $N_{\text{pair}} = N(N - 1)/2$ columns are ordered: $C_{12}, C_{13}, \dots, C_{1N}, C_{23}, \dots, C_{2N}, C_{34}, \dots, C_{N-1,N}$
- For *type = lower*, the $N_{\text{pair}} = N(N - 1)/2$ columns are ordered: $C_{21}, C_{31}, C_{32}, C_{41}, C_{42}, C_{43}, \dots, C_{N1}, C_{N2}, \dots, C_{N,N-1}$
- For *type = auto/upper*, the $N_{\text{pair}} = N(N + 1)/2$ columns are ordered: $C_{11}, C_{12}, C_{13}, \dots, C_{1N}, C_{22}, C_{23}, \dots, C_{2N}, C_{33}, C_{34}, \dots, C_{N-1,N}, C_{NN}$
- For *type = auto/lower*, the $N_{\text{pair}} = N(N + 1)/2$ columns are ordered: $C_{11}, C_{21}, C_{22}, C_{31}, C_{32}, C_{33}, C_{41}, \dots, C_{44}, C_{N1}, C_{N2}, \dots, C_{N,N-1}, C_{NN}$
- For *type = full*, the $N_{\text{pair}} = N^2$ columns are ordered: $C_{11}, C_{12}, \dots, C_{1N}, C_{21}, C_{22}, \dots, C_{2N}, C_{31}, \dots, C_{3N}, \dots, C_{N1}, \dots, C_{N,N-1}, C_{NN}$

The array values calculated by this fix are treated as extensive. 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*.

2.16.5 Restrictions

none

2.16.6 Related commands

fix ave/correlate/long, compute, fix ave/time, fix ave/atom, fix ave/chunk, fix ave/histo, variable

2.16.7 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.

2.17 fix ave/correlate/long command

2.17.1 Syntax

```
fix ID group-ID ave/correlate/long Nevery Nfreq value1 value2 ... keyword args ...
```

- ID, group-ID are documented in [fix](#) command
- ave/correlate/long = style name of this fix command
- Nevery = use input values every this many time steps
- Nfreq = save state of the time correlation functions every this many time steps
- one or more input values can be listed
- value = c_ID, c_ID[N], f_ID, f_ID[N], v_name, v_name[I]

c_ID = global scalar calculated by a compute with ID
c_ID[I] = Ith component of global vector calculated by a compute with ID, I can include wildcard [_](#) (see below)
f_ID = global scalar calculated by a fix with ID
f_ID[I] = Ith component of global vector calculated by a fix with ID, I can include wildcard [_](#) (see below)
v_name = global value calculated by an equal-style variable with name
v_name[I] = Ith component of a vector-style variable with name, I can include wildcard [_](#) (see below)

- zero or more keyword/arg pairs may be appended
- keyword = *type* or *start* or *file* or *overwrite* or *title1* or *title2* or *ncorr* or *nlen* or *ncount*

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 + lower

start args = Nstart

Nstart = start accumulating correlations on this time step

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
ncorr arg = Ncorrelators
Ncorrelators = number of correlators to store
nlen args = Nlen
Nlen = length of each correlator
ncount args = Ncount
Ncount = number of values over which successive correlators are averaged

```

2.17.2 Examples

```

fix 1 all ave/correlate/long 5 1000 c_myTemp file temp.correlate
fix 1 all ave/correlate/long 1 10000 &
  c_thermo_press[1] c_thermo_press[2] c_thermo_press[3] &
  type upper title1 "My correlation data" nlen 15 ncount 3
fix 1 all ave/correlate/long 1 10000 c_thermo_press[*]

```

2.17.3 Description

This fix is similar in spirit and syntax to the [fix ave/correlate](#). However, this fix allows the efficient calculation of time correlation functions on-the-fly over extremely long time windows with little additional CPU overhead, using a multiple- τ method ([Ramirez](#)) that decreases the resolution of the stored correlation function with time. It is not a full drop-in replacement.

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 or vector-style variable. For vector-style variables, the specified indices can include a wildcard character. See the [fix ave/correlate](#) page for details.

The *Nevery* and *Nfreq* arguments specify on what time steps the input values will be used to calculate correlation data and the frequency with which the time correlation functions will be output to a file, respectively. Note that there is no *Nrepeat* argument, unlike the [fix ave/correlate](#) command.

The optional keywords *ncorr*, *nlen*, and *ncount* are unique to this command and determine the number of correlation points calculated and the memory and CPU overhead used by this calculation. *Nlen* and *ncount* determine the amount of averaging done at longer correlation times. The default values *nlen* = 16 and *ncount* = 2 ensure that the systematic error of the multiple- τ correlator is always below the level of the statistical error of a typical simulation (which depends on the ensemble size and the simulation length).

The maximum correlation time (in time steps) that can be reached is given by the formula $(nlen - 1)ncount^{(ncorr-1)}$. Longer correlation times are discarded and not calculated. With the default values of the parameters (*ncorr* = 20, *nlen* = 16 and *ncount* = 2), this corresponds to 7864320 time steps. If longer correlation times are needed, the value of *ncorr* should be increased. Using *nlen* = 16 and *ncount* = 2, with *ncorr* = 30, the maximum number of steps that can be correlated is 80530636808. If *ncorr* = 40, correlation times in excess of 8×10^{12} time steps can be calculated.

The total memory needed for each correlation pair is roughly $4 \times ncorr \times nlen \times 8$ bytes. With the default values of the parameters, this corresponds to about 10 KB.

For the meaning of the additional optional keywords, see the [fix ave/correlate](#) doc page.

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

Contrary to [fix ave/correlate](#) this fix does **not** provide access to its internal data to various output options. Since this fix is intended for the calculation of time correlation functions over very long MD simulations, the information about this fix is written automatically to binary restart files, so that the time correlation calculation can continue in subsequent simulations. 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.

2.17.5 Restrictions

This compute is part of the EXTRA-FIX package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

2.17.6 Related commands

[fix ave/correlate](#)

2.17.7 Default

none

The option defaults for keywords that are also keywords for the [fix ave/correlate](#) command are as follows: type = auto, start = 0, no file output, title 1,2 = strings as described on the [fix ave/correlate](#) doc page.

The option defaults for keywords unique to this command are as follows: ncorr=20, nlen=16, ncount=2.

(Ramirez) J. Ramirez, S.K. Sukumaran, B. Vorselaars and A.E. Likhtman, *J. Chem. Phys.* 133, 154103 (2010).

2.18 fix ave/grid command

2.18.1 Syntax

```
fix ID group-ID ave/grid Nevery Nrepeat Nfreq Nx Ny Nz value1 value2 ... keyword args ...
```

- ID, group-ID are documented in [fix](#) command
- ave/grid = 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
- Nx, Ny, Nz = grid size in each dimension
- one or more per-atom or per-grid input values can be listed
- per-atom value = vx, vy, vz, fx, fy, fz, density/mass, density/number, mass, temp, c_ID, c_ID[I], f_ID, f_ID[I], v_name

vx,vy,vz,fx,fz,mass = atom attribute (velocity, force component, mass)
 density/number, density/mass = number or mass density (per volume)
 temp = temperature
 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, I can include wildcard [_](#)
 ↵(see below)
 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, I can include wildcard [_](#)
 ↵(see below)
 v_name = per-atom vector calculated by an atom-style variable with name

- per-grid value = c_ID:gname:dname, c_ID:gname:dname[I], f_ID:gname:dname, f_ID:gname:dname[I]

gname = name of grid defined by compute or fix
 dname = name of data field defined by compute or fix
 c_ID = per-grid vector calculated by a compute with ID
 c_ID[I] = Ith column of per-grid array calculated by a compute with ID, I can include wildcard [_](#)
 ↵(see below)
 f_ID = per-grid vector calculated by a fix with ID
 f_ID[I] = Ith column of per-grid array calculated by a fix with ID, I can include wildcard [_](#)
 ↵(see below)

- zero or more keyword/arg pairs may be appended
- keyword = *discard* or *norm* or *ave* or *bias* or *adof* or *c dof*

discard arg = yes or no
 yes = discard an atom outside grid in a non-periodic dimension
 no = remap an atom outside grid in a non-periodic dimension to first or last grid cell
 norm arg = all or sample or none = how output on Nfreq steps is normalized
 all = output is sum of atoms across all Nrepeat samples, divided by atom count
 sample = output is sum of Nrepeat sample averages, divided by Nrepeat
 none = output is sum of Nrepeat sample sums, divided by Nrepeat
 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
 bias arg = bias-ID
 bias-ID = ID of a temperature compute that removes a velocity bias for temperature calculation
 adof value = dof_per_atom
 dof_per_atom = define this many degrees-of-freedom per atom for temperature calculation
 cdof value = dof_per_grid_cell
 dof_per_grid_cell = add this many degrees-of-freedom per grid_cell for temperature calculation

2.18.2 Examples

```
fix 1 all ave/grid 10000 1 10000 10 10 10 fx fy fz c_myMSD[*]
fix 1 flow ave/grid 100 10 1000 20 20 30 f_TTM:grid:data
```

2.18.3 Description

Overlay the 2d or 3d simulation box with a uniformly spaced 2d or 3d grid and use it to either (a) time-average per-atom quantities for the atoms in each grid cell, or to (b) time-average per-grid quantities produced by other computes or fixes. This fix operates in either “per-atom mode” (all input values are per-atom) or in “per-grid mode” (all input values are per-grid). You cannot use both per-atom and per-grid inputs in the same command.

The grid created by this command is distributed; each processor owns the grid points that are within its subdomain. This is similar to the [fix ave/chunk](#) command when it uses chunks from the [compute chunk/atom](#) command which are 2d or 3d regular bins. However, the per-bin outputs in that case are global; each processor stores a copy of the entire set of bin data. Thus it more efficient to use the fix ave/grid command when the grid is large and a simulation is run on many processors.

For per-atom mode, only atoms in the specified group contribute to the summing and averaging calculations. For per-grid mode, the specified group is ignored.

The N_{every} , N_{repeat} , and N_{freq} arguments specify on what time steps the input values will be accessed and contribute to the average. The final averaged quantities are generated on time steps that are multiples of N_{freq} . The average is over N_{repeat} quantities, computed in the preceding portion of the simulation every N_{every} time steps. N_{freq} must be a multiple of N_{every} and N_{every} must be non-zero even if $N_{\text{repeat}} = 1$. Also, the time steps contributing to the average value cannot overlap (i.e., $N_{\text{repeat}} \times N_{\text{every}}$ cannot exceed N_{freq}).

For example, if $N_{\text{every}} = 2$, $N_{\text{repeat}} = 6$, and $N_{\text{freq}} = 100$, then values on time steps 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 $N_{\text{repeat}} = 1$ and $N_{\text{freq}} = 100$, then no time averaging is done; values are simply generated on timesteps 100,200,etc.

In per-atom mode, each input value can also be averaged over the atoms in each grid cell. The way the averaging is done across the N_{repeat} timesteps to produce output on the N_{freq} timesteps, and across multiple N_{freq} outputs, is determined by the *norm* and *ave* keyword settings, as discussed below.

The N_x , N_y , and N_z arguments specify the size of the grid that overlays the simulation box. For 2d simulations, N_z must be 1. The N_x , N_y , N_z values can be any positive integer. The grid can be very coarse compared to the particle count, or very fine. If one or more of the values = 1, then bins are 2d planes or 1d slices of the simulation domain. Note that if the total number of grid cells is small, it may be more efficient to use the [fix ave/chunk](#) command which can treat a grid defined by the [compute chunk/atom](#) command as a global grid where each processor owns a copy of all the grid cells. If $N_x = N_y = N_z = 1$ is used, the same calculation would be more efficiently performed by the [fix ave/atom](#) command.

If the simulation box size or shape changes during a simulation, the grid always conforms to the size/shape of the current simulation box. If one more dimensions have non-periodic shrink-wrapped boundary conditions, as defined by the [boundary](#) command, then the grid will extend over the (dynamic) shrink-wrapped extent in each dimension. If the box shape is triclinic, as explained in [Howto triclinic](#), then the grid is also triclinic; each grid cell is a small triclinic cell with the same shape as the simulation box.

In both per-atom and per-grid mode, input values from a compute or fix that produces an array of values (multiple values per atom or per grid point), the bracketed index I can be specified using a wildcard asterisk with the index to effectively specify multiple values. This takes the form “*” or “*n” or “n*” or “m*n”. If N = the number of columns in

the array (for *mode* = vector), then an asterisk with no numeric values means all indices from 1 to N. A leading asterisk means all indices from 1 to n (inclusive). A trailing asterisk means all indices from n to N (inclusive). A middle asterisk means all indices from m to n (inclusive).

Using a wildcard is the same as if the individual columns of the array had been listed one by one. E.g. if there were a compute fft/grid command which produced 3 values for each grid point, these two fix ave/grid commands would be equivalent:

```
compute myFFT all fft/grid 10 10 10 ...
fix 1 all ave/grid 100 1 100 10 10 10 c_myFFT:grid:data[*]
fix 2 all ave/grid 100 1 100 10 10 10 c_myFFT:grid:data[*][1] c_myFFT:grid:data[*][2] c_
-myFFT:grid:data[3]
```

Per-atom mode:

Each specified per-atom value can be an atom attribute (velocity, force component), a number or mass density, a mass or temperature, 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. Note that the [compute property/atom](#) command provides access to any attribute defined and stored by atoms.

The per-atom values of each input vector are summed and averaged independently of the per-atom values in other input vectors.

[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 atom attribute values (vx,vy,vz,fx,fy,fz,mass) are self-explanatory. As noted above, any other atom attributes can be used as input values 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 for each grid cell, i.e. number/volume. The *density/mass* value means the mass density is computed for each grid/cell, i.e. total-mass/volume. The output values are in units of 1/volume or density (mass/volume). See the [units](#) command page for the definition of density for each choice of units, e.g. gram/cm³.

The *temp* value computes the temperature for each grid cell, by the formula

$$\text{KE} = \frac{\text{DOF}}{2} k_B T,$$

where KE = total kinetic energy of the atoms in the grid cell ($\frac{1}{2}mv^2$), DOF = the total number of degrees of freedom for all atoms in the grid cell, k_B = Boltzmann constant, and T = temperature.

The DOF is calculated as N*adof + cdof, where N = number of atoms in the grid cell, adof = degrees of freedom per atom, and cdof = degrees of freedom per grid cell. By default adof = 2 or 3 = dimensionality of system, as set via the [dimension](#) command, and cdof = 0.0. This gives the usual formula for temperature.

Note that currently this temperature only includes translational degrees of freedom for each atom. No rotational degrees of freedom are included for finite-size particles. Also no degrees of freedom are subtracted for any velocity bias or constraints that are applied, such as [compute temp/partial](#), or [fix shake](#) or [fix rigid](#). This is because those degrees of freedom (e.g. a constrained bond) could apply to sets of atoms that are both inside and outside a specific grid cell, and hence the concept is somewhat ill-defined. In some cases, you can use the *adof* and *cdof* keywords to adjust the calculated degrees of freedom appropriately, as explained below.

Also note that a bias can be subtracted from atom velocities before they are used in the above formula for KE, by using the *bias* keyword. This allows, for example, a thermal temperature to be computed after removal of a flow velocity profile.

Note that the per-grid-cell temperature calculated by this fix and the [compute temp/chunk](#) command (using bins) can be different. The compute calculates the temperature for each chunk for a single snapshot. This fix can do that but can also time average those values over many snapshots, or it can compute a temperature as if the atoms in the grid cell on different timesteps were collected together as one set of atoms to calculate their temperature. The compute allows the center-of-mass velocity of each chunk to be subtracted before calculating the temperature; this fix does not.

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 LAMMPS](#). See the discussion above for how I can be specified with a wildcard asterisk to effectively specify multiple values.

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 N_{every} , else an error results. Users can also write code for their own fix styles and [add them to LAMMPS](#). See the discussion above for how I can be specified with a wildcard asterisk to effectively specify multiple values.

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 average within grid cells.

Per-grid mode:

The attributes that begin with *c_ID* and *f_ID* both take colon-separated fields *gname* and *dname*. These refer to a grid name and data field name which is defined by the compute or fix. Note that a compute or fix can define one or more grids (of different sizes) and one or more data fields for each of those grids. The sizes of all grids used as values for one instance of this fix must be the same.

The *c_ID:gname:dname* and *c_ID:gname:dname[I]* attributes allow per-grid vectors or arrays calculated by a [compute](#) to be accessed. The ID in the attribute should be replaced by the actual ID of the compute that has been defined previously in the input script.

If *c_ID:gname:dname* is used as a attribute, then the per-grid vector calculated by the compute is accessed. If *c_ID:gname:dname[I]* is used, then I must be in the range from 1-M, which will access the Ith column of the per-grid array with M columns calculated by the compute. See the discussion above for how I can be specified with a wildcard asterisk to effectively specify multiple values.

The *f_ID:gname:dname* and *f_ID:gname:dname[I]* attributes allow per-grid 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:gname:dname* is used as a attribute, then the per-grid vector calculated by the fix is printed. If *f_ID:gname:dname[I]* is used, then I must be in the range from 1-M, which will print the Ith column of the per-grid with M columns calculated by the fix. See the discussion above for how I can be specified with a wildcard asterisk to effectively specify multiple values.

Additional optional keywords also affect the operation of this fix and its outputs. Some are only applicable to per-atom mode. Some are applicable to both per-atom and per-grid mode.

The *discard* keyword is only applicable to per-atom mode. If a dimension of the system is non-periodic, then grid cells will only span the box dimension (fixed or shrink-wrap boundaries as set by the [boundary command](#) command). An atom may thus be slightly outside the range of grid cells on a particular timestep. If *discard* is set to yes (the default),

then the atom will be assigned to the closest grid cell (lowest or highest) in that dimension. If *discard* is set to *no* the atom will be ignored.

The *norm* keyword is only applicable to per-atom mode. In per-grid mode, the *norm* keyword setting is ignored. The output grid value on an N_{freq} timestep is the sum of the grid values in each of the N_{repeat} samples, divided by N_{repeat} .

In per-atom mode, the *norm* keyword affects how averaging is done for the per-grid values that are output on an N_{freq} timestep. N_{repeat} samples contribute to the output. The *norm* keyword has 3 possible settings: *all* or *sample* or *none*. *All* is the default.

In the formulas that follow, SumI is the sum of a per-atom property over the CountI atoms in a grid cell for a single sample I, where I varies from 1 to N, and N = N_{repeat} . These formulas are used for any per-atom input value listed above, except *density/number*, *density/mass*, and *temp*. Those input values are discussed below.

In per-atom mode, for *norm all* the output grid value on the N_{freq} timestep is an average over atoms across the entire N_{freq} timescale:

$$\text{Output} = (\text{Sum1} + \text{Sum2} + \dots + \text{SumN}) / (\text{Count1} + \text{Count2} + \dots + \text{CountN})$$

In per-atom mode, for *norm sample* the output grid value on the N_{freq} timestep is an average of an average:

$$\text{Output} = (\text{Sum1}/\text{Count1} + \text{Sum2}/\text{Count2} + \dots + \text{SumN}/\text{CountN}) / N_{\text{repeat}}$$

In per-atom mode, for *norm none* the output grid value on the N_{freq} timestep is not normalized by the atom counts:

$$\text{Output} = (\text{Sum1} + \text{Sum2} + \dots + \text{SumN}) / N_{\text{repeat}}$$

For *density/number* and *density/mass*, the output value is the same as in the formulas above for *norm all* and *norm sample*, except that the result is also divided by the grid cell volume. For *norm all*, this will be the volume at the final N_{freq} timestep. For *norm sample*, the divide-by-volume is done for each sample, using the grid cell volume at the sample timestep. For *norm none*, the output is the same as for *norm all*.

For *temp*, the output temperature uses the formula for kinetic energy KE listed above, and is normalized similarly to the formulas above for *norm all* and *norm sample*, except for the way the degrees of freedom (DOF) are calculated. For *norm none*, the output is the same as for *norm all*.

For *norm all*, the $\text{DOF} = N_{\text{repeat}} \times \text{cdof}$ plus Count times adof , where $\text{Count} = (\text{Count1} + \text{Count2} + \dots + \text{CountN})$. The *cdof* and *adof* keywords are discussed below. The output temperature is computed with all atoms across all samples contributing.

For *norm sample*, the DOF for a single sample = *cdof* plus *Count* times *adof*, where *Count* = *CountI* for a single sample. The output temperature is the average of N_{sample} temperatures calculated for each sample.

Finally, for all 3 *norm* settings the output count of atoms per grid cell is:

$$\text{Output count} = (\text{Count1} + \text{Count2} + \dots + \text{CountN}) / N_{\text{repeat}}$$

This count is the same for all per-atom input values, including *density/number*, *density/mass*, and *temp*.

The *ave* keyword is applied to both per-atom and per-grid mode. It determines how the per-grid values produced once every N_{freq} steps are averaged with values produced on previous steps that were multiples of N_{freq} , before they are accessed by another output command.

If the *ave* setting is *one*, which is the default, then the grid values produced on N_{freq} timesteps are independent of each other; they are output as-is without further averaging.

If the *ave* setting is *running*, then the grid values produced on N_{freq} timesteps are summed and averaged in a cumulative sense before being output. Each output grid value is thus the average of the grid value produced on that timestep with all preceding values for the same grid value. 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 grid values produced on N_{freq} timesteps are summed and averaged within a moving “window” of time, so that the last M values for the same grid are used to produce the output. E.g. if M = 3 and Nfreq = 1000, then the grid value output on step 10000 will be the average of the grid values on steps 8000,9000,10000. Outputs on early steps will average over less than M values if they are not available.

The *bias*, *adof*, and *c dof* keywords are only applicable to per-atom mode.

The *bias* keyword specifies the ID of a temperature compute that removes a “bias” velocity from each atom, specified as *bias-ID*. It is only used when the *temp* value is calculated, to compute the thermal temperature of each grid cell after the translational kinetic energy components have been altered in a prescribed way, e.g. to remove a flow velocity profile. See the doc pages for individual computes that calculate a temperature to see which ones implement a bias.

The *adof* and *c dof* keywords define the values used in the degree of freedom (DOF) formula described above for temperature calculation for each grid cell. They are only used when the *temp* value is calculated. They can be used to calculate a more appropriate temperature in some cases. Here are 3 examples:

If grid cells contain some number of water molecules and *fix shake* is used to make each molecule rigid, then you could calculate a temperature with 6 degrees of freedom (DOF) (3 translational, 3 rotational) per molecule by setting *adof* to 2.0.

If *compute temp/partial* is used with the *bias* keyword to only allow the x component of velocity to contribute to the temperature, then *adof* = 1.0 would be appropriate.

Using *c dof* = -2 or -3 (for 2d or 3d simulations) will subtract out 2 or 3 degrees of freedom for each grid cell, similar to how the *compute temp* command subtracts out 3 DOF for the entire system.

2.18.4 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 calculates a per-grid array which has one column for each of the specified input values. The units for each column will be in the *units* for the per-atom or per-grid quantity for the corresponding input value. If the fix is used in per-atom mode, it also calculates a per-grid vector with the count of atoms in each grid cell. The number of rows in the per-grid array and number of values in the per-grid vector (distributed across all processors) is $N_x * N_y * N_z$.

For access by other commands, the name of the single grid produced by this fix is “grid”. The names of its two per-grid datums are “data” for the per-grid array and “count” for the per-grid vector (if using per-atom values). Both datums can be accessed by various *output commands*.

In per-atom mode, the per-grid array values calculated by this fix are treated as “intensive”, since they are typically already normalized by the count of atoms in each grid cell.

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*.

2.18.5 Restrictions

none

2.18.6 Related commands

fix ave/atom, *fix ave/chunk*

2.18.7 Default

The option defaults are discard = yes, norm = all, ave = one, and bias = none.

2.19 fix ave/histo command

2.20 fix ave/histo/weight command

2.20.1 Syntax

`fix ID group-ID style Nevery Nrepeat Nfreq lo hi Nbin value1 value2 ... keyword args ...`

- ID, group-ID are documented in *fix* command
- style = *ave/histo* or *ave/histo/weight* = 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, I can include wildcard (see below)
 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, I can include wildcard (see below)
 v_name = value(s) calculated by an equal-style or vector-style or atom-style variable with name
 v_name[I] = value calculated by a vector-style variable with name, I can include wildcard (see below)

- zero or more keyword/arg pairs may be appended
- keyword = *mode* or *kind* or *file* or *append* 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

kind arg = global or peratom or local

file arg = filename

filename = name of file to output histogram(s) to

```

append arg = filename
  filename = name of file to append 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

```

2.20.2 Examples

```

fix 1 all ave/histo 100 5 1000 0.5 1.5 50 c_myTemp file temp.histo ave running
fix 1 all ave/histo 100 5 1000 -5 5 100 c_thermo_press[2] c_thermo_press[3] title1 "My output values"
fix 1 all ave/histo 100 5 1000 -5 5 100 c_thermo_press[*]
fix 1 all ave/histo 1 100 1000 -2.0 2.0 18 vx vy vz mode vector ave running beyond extra
fix 1 all ave/histo/weight 1 1 1 10 100 2000 c_XRD[1] c_XRD[2]

```

2.20.3 Description

Use one or more values as inputs every few timesteps to create a single histogram. The histogram can then be averaged over longer timescales. The resulting histogram can be used by other [output commands](#), and can also be written to a file. The fix ave/histo/weight command has identical syntax to fix ave/histo, except that exactly two values must be specified. See details below.

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 vector-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 page for individual “compute” and “fix” commands to see what kinds of quantities they generate.

Note that a compute or fix can produce multiple kinds of data (global, per-atom, local). If LAMMPS cannot unambiguously determine which kind of data to use, the optional *kind* keyword discussed below can force the desired disambiguation.

Note that 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 input values must either be all scalars or all vectors (or arrays), depending on the setting of the *mode* keyword.

If *mode* = scalar, then the input values must be scalars, or vectors with a bracketed term appended, indicating the *I*th value of the vector is used.

If *mode* = vector, then the input values must be vectors, or arrays with a bracketed term appended, indicating the *I*th column of the array is used.

If the fix ave/histo/weight command is used, exactly two values must be specified. If the values are vectors, they must be the same length. The first value (a scalar or vector) is what is histogrammed into bins, in the same manner the fix ave/histo command operates. The second value (a scalar or vector) is used as a “weight”. This means that instead of each value tallying a “1” to its bin, the corresponding weight is tallied. For example, the N^{th} entry (weight) in the second vector is tallied to the bin corresponding to the N^{th} entry in the first vector.

For input values from a compute or fix or variable, the bracketed index *I* can be specified using a wildcard asterisk with the index to effectively specify multiple values. This takes the form “*” or “**n*” or “*m**” or “*m***n*”. If *N* is the size of the vector (for *mode* = scalar) or the number of columns in the array (for *mode* = vector), then an asterisk with no numeric values means all indices from 1 to *N*. A leading asterisk means all indices from 1 to *n* (inclusive). A trailing asterisk means all indices from *m* to *N* (inclusive). A middle asterisk means all indices from *m* to *n* (inclusive).

Using a wildcard is the same as if the individual elements of the vector or columns of the array had been listed one by one. For example, the following two fix ave/histo commands are equivalent, since the *compute com/chunk* command creates a global array with three columns:

```
compute myCOM all com/chunk
fix 1 all ave/histo 100 1 100 -10.0 10.0 100 c_myCOM[*] file tmp1.com mode vector
fix 2 all ave/histo 100 1 100 -10.0 10.0 100 c_myCOM[1] c_myCOM[2] c_myCOM[3] file tmp2.com
  mode vector
```

Note

For a vector-style variable, only the wildcard forms “**n*” or “*m***n*” are allowed. You must specify the upper bound, because vector-style variable lengths are not determined until the variable is evaluated. If *n* is specified larger than the vector length turns out to be, zeroes are output for missing vector values.

The *Nevery*, *Nrepeat*, and *Nfreq* arguments specify on what time steps the input values will be used in order to contribute to the histogram. The final histogram is generated on time steps that are multiple of *Nfreq*. It is averaged over *Nrepeat* histograms, computed in the preceding portion of the simulation every *Nevery* time steps. *Nfreq* must be a multiple of *Nevery* and *Nevery* must be non-zero even if *Nrepeat* is 1. Also, the time steps contributing to the histogram value cannot overlap (i.e., $N_{\text{repeat}} \times N_{\text{every}}$ cannot exceed *Nfreq*).

For example, if *Nevery* = 2, *Nrepeat* = 6, and *Nfreq* = 100, then input values on time steps 90, 92, 94, 96, 98, and 100 will be used to compute the final histogram on timestep 100. Similarly for timesteps 190, 192, 194, 196, 198, and 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*, , and *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 *I*th 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. If a bracketed term is appended, the *I*th column of the global or per-atom or local array calculated by the compute is used. See the discussion above for how *I* can be specified with a wildcard asterisk to effectively specify multiple values.

Note that there is a [compute reduce](#) command that can sum per-atom quantities into a global scalar or vector, which can then be accessed by fix ave/histo. It can also be a compute defined not in your input script, but by [thermodynamic output](#) or other fixes such as [fix nvt](#) or [fix temp/rescale](#). See the doc pages for these commands which give the IDs of these computes. Users can also write code for their own compute styles and [add them to LAMMPS](#).

If a value begins with “f_”, a fix ID must follow which has been previously defined in the input script. If *mode* = scalar, then if no bracketed term is appended, the global scalar calculated by the fix is used. If a bracketed term is appended, the *I*th 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. If a bracketed term is appended, the *I*th column of the global or per-atom or local array calculated by the fix is used. See the discussion above for how *I* can be specified with a wildcard asterisk to effectively specify multiple values.

Note that some fixes only produce their values on certain timesteps, which must be compatible with *Nevery*, else an error will result. Users can also write code for their own fix styles and [add them to LAMMPS](#).

If a value begins with “v_”, a variable name must follow which has been previously defined in the input script. If *mode* = scalar, then only equal-style or vector-style variables can be used, which both produce global values. In this mode, a vector-style variable requires a bracketed term to specify the *I*th element of the vector calculated by the variable. If *mode* = vector, then only vector-style or atom-style variables can be used, which produce a global or per-atom vector respectively. The vector-style variable must be used without a bracketed term. See the [variable](#) command for details.

Note that variables of style *equal*, *vector*, 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 *kind* keyword only needs to be used if any of the specified input computes or fixes produce more than one kind of output (global, per-atom, local). If not, LAMMPS will determine the kind of data all the inputs produce and verify it is all the same kind. If not, an error will be triggered. If a compute or fix produces more than one kind of output, the *kind* keyword should be used to specify which output will be used. The other input arguments must still be consistent.

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 two 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* + 2). Values between *lo* and *hi* (inclusive) are counted in bins 2 through *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 N_{freq} 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 $N_{\text{freq}} = 1000$, then the output on step 10000 will be the combined histogram of the individual histograms on steps 8000, 9000, and 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.

Added in version 17Apr2024: new keyword *append*

The *file* or *append* keywords allow a filename to be specified. 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. Every N_{freq} 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*, *title2*, and *title3* keywords allow specification of the strings that will be printed as the first three lines of the output file, assuming the *file* keyword was used. LAMMPS uses default values for each of these, so they do not need to be specified.

By default, these header lines are as follows:

```
# Histogram for fix ID
# TimeStep Number-of-bins Total-counts Missing-counts Min-value Max-value
# Bin Coord Count Count/Total
```

In the first line, ID is replaced with the fix-ID. The second line describes the six values that are printed at the first of each section of output. The third describes the four values printed for each bin in the histogram.

2.20.4 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 N_{freq} since that is when a histogram is generated. The global vector has four 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 N_{bins} rows and three columns. The first column has the bin coordinate, the second column has the count of values in that histogram bin, and the third column has the bin count divided by the total count (not including missing counts), so that the values in the third column sum to 1.0.

The vector and array values calculated by this fix are all treated as intensive. If this is not the case (e.g., due to histogramming per-atom input values), then you will need to account for that when interpreting the values produced by this fix.

No parameter of this fix can be used with the *start/stop* keywords of the [run](#) command. This fix is not invoked during *energy minimization*.

2.20.5 Restrictions

none

2.20.6 Related commands

[compute](#), [fix ave/atom](#), [fix ave/chunk](#), [fix ave/time](#), [variable](#), [fix ave/correlate](#),

2.20.7 Default

none

The option defaults are mode = scalar, kind = figured out from input arguments, ave = one, start = 0, no file output, beyond = ignore, and title 1,2,3 = strings as described above.

2.21 fix ave/spatial command

Deprecated since version 11Dec2015.

The *fix ave/spatial* command has been superseded by [fix ave/chunk](#).

2.22 fix ave/spatial/sphere command

Deprecated since version 11Dec2015.

The *fix ave/spatial/sphere* command has been superseded by [fix ave/chunk](#).

2.23 fix ave/time command

2.23.1 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 time steps

- Nrepeat = # of times to use input values for calculating averages
- Nfreq = calculate averages every this many time steps
- 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 compute with ID, I can include wildcard (see below)
f_ID = global scalar or vector calculated by a fix with ID
f_ID[I] = Ith component of global vector or Ith column of global array calculated by a fix with ID, I can include wildcard (see below)
v_name = value(s) calculated by an equal-style or vector-style variable with name
v_name[I] = value calculated by a vector-style variable with name, I can include wildcard (see below)

- zero or more keyword/arg pairs may be appended
- keyword = *mode* or *file* or *append* or *ave* or *start* or *off* or *overwrite* or *format* 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 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 time step

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

append arg = filename

filename = name of file to append time averages to

overwrite arg = none = overwrite output file with only latest output

format arg = string

string = C-style format string

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

2.23.2 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 100 5 1000 c_thermo_press[*]
fix 1 all ave/time 1 100 1000 f_indent f_indent[1] file temp.indent off 1
```

2.23.3 Description

Use one or more global values as inputs every few time steps, 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 or vector-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/chunk*, *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* and *vector* 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 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 or each element of each input vector is averaged independently).

If *mode* = scalar, then the input values must be scalars, or vectors with a bracketed term appended, indicating the *I*th value of the vector is used.

If *mode* = vector, then the input values must be vectors, or arrays with a bracketed term appended, indicating the *I*th column of the array is used. All vectors must be the same length, which is the length of the vector or number of rows in the array.

For input values from a compute or fix or variable, the bracketed index *I* can be specified using a wildcard asterisk with the index to effectively specify multiple values. This takes the form “*” or “*n” or “m*” or “m*n”. If *N* is the size of the vector (for *mode* = scalar) or the number of columns in the array (for *mode* = vector), then an asterisk with no numeric values means all indices from 1 to *N*. A leading asterisk means all indices from 1 to *n* (inclusive). A trailing asterisk means all indices from *n* to *N* (inclusive). A middle asterisk means all indices from *m* to *n* (inclusive).

Using a wildcard is the same as if the individual elements of the vector or columns of the array had been listed one by one. For example, the following two fix ave/time commands are equivalent, since the *compute rdf* command creates, in this case, a global array with three 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
```

Note

For a vector-style variable, only the wildcard forms “*n” or “m*n” are allowed. You must specify the upper bound, because vector-style variable lengths are not determined until the variable is evaluated. If n is specified larger than the vector length turns out to be, zeroes are output for missing vector values.

The N_{every} , N_{repeat} , and N_{freq} arguments specify on what time steps the input values will be used in order to contribute to the average. The final averaged quantities are generated on time steps that are a multiple of N_{freq} . The average is over N_{repeat} quantities, computed in the preceding portion of the simulation every N_{every} time steps. N_{freq} must be a multiple of N_{every} and N_{every} must be non-zero even if $N_{\text{repeat}} = 1$. Also, the time steps contributing to the average value cannot overlap (i.e., $N_{\text{repeat}} \times N_{\text{every}}$ cannot exceed N_{freq}).

For example, if $N_{\text{every}} = 2$, $N_{\text{repeat}} = 6$, and $N_{\text{freq}} = 100$, then values on time steps 90, 92, 94, 96, 98, and 100 will be used to compute the final average on time step 100. Similarly for time steps 190, 192, 194, 196, 198, and 200 on time step 200, etc. If $N_{\text{repeat}} = 1$ and $N_{\text{freq}} = 100$, then no time averaging is done; values are simply generated on time steps 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. If a bracketed term is appended, the Ith column of the global array calculated by the compute is used. See the discussion above for how I can be specified with a wildcard asterisk to effectively specify multiple values.

Note that there is a *compute reduce* command that can sum per-atom quantities into a global scalar or vector, which can then be accessed by fix ave/time. It can also be a compute defined not in your input script, but by *thermodynamic output* or other fixes such as *fix nvt* or *fix temp/rescale*. See the doc pages for these commands which give the IDs of these computes. Users can also write code for their own compute styles and *add them to LAMMPS*.

If a value begins with “f_”, a fix ID must follow which has been previously defined in the input script. If *mode* = scalar, then if no bracketed term is appended, the global scalar calculated by the fix is used. If a bracketed term is appended, the Ith element of the global vector calculated by the fix is used. If *mode* = vector, then if no bracketed term is appended, the global vector calculated by the fix is used. If a bracketed term is appended, the Ith column of the global array calculated by the fix is used. See the discussion above for how I can be specified with a wildcard asterisk to effectively specify multiple values.

Note that some fixes only produce their values on certain time steps, which must be compatible with *Nevery*, else an error will result. Users can also write code for their own fix styles and *add them to LAMMPS*.

If a value begins with “v_”, a variable name must follow which has been previously defined in the input script. If *mode* = scalar, then only equal-style or vector-style variables can be used, which both produce global values. In this mode, a vector-style variable requires a bracketed term to specify the Ith element of the vector calculated by the variable. If *mode* = vector, then only a vector-style variable can be used, without a bracketed term. See the *variable* command for details.

Note that variables of style *equal* and *vector* 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 N_{freq} steps are averaged with values produced on previous steps that were multiples of N_{freq} , before they are accessed by another output command or written to a file.

If the *ave* setting is *one*, then the values produced on time steps that are multiples of N_{freq} are independent of each other; they are output as-is without further averaging.

If the *ave* setting is *running*, then the values produced on time steps that are multiples of N_{freq} are summed and averaged in a cumulative sense before being output. Each output value is thus the average of the value produced on that time step 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 time steps that are multiples of N_{freq} are summed and averaged within a moving “window” of time, so that the last M values are used to produce the output. For example, if $M = 3$ and $N_{\text{freq}} = 1000$, then the output on step 10000 will be the average of the individual values on steps 8000, 9000, and 10000. Outputs on early steps will average over less than M values if they are not available.

The *start* keyword specifies what time step 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 or 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).

Added in version 17Apr2024: new keyword *append*

The *file* or *append* keywords allow a filename to be specified. 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. Every N_{freq} 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.

Added in version 4May2022.

If the filename ends in ‘.yaml’ or ‘.yml’ then the output format conforms to the [YAML](#) standard which allows easy import that data into tools and scripts that support reading YAML files. The [structured data Howto](#) contains examples for parsing and plotting such data with very little programming effort in Python using the *pyyaml*, *pandas*, and *matplotlib* packages.

The *overwrite* keyword will continuously overwrite the output file with the latest output, so that it only contains one time step worth of output. This option can only be used with the *ave running* setting.

The *format* keyword sets the numeric format of each value when it is printed to a file via the *file* keyword. Note that all values are floating point quantities. The default format is `%g`. You can specify a higher precision if desired (e.g., `%20.16g`).

The *title1* and *title2* and *title3* keywords allow specification of the strings that will be printed as the first 2 or 3 lines of the output file, assuming the *file* keyword was used. LAMMPS uses default values for each of these, so they do not need to be specified.

By default, these header lines are as follows for *mode* = scalar:

```
# Time-averaged data for fix ID  
# TimeStep value1 value2 ...
```

In the first line, ID is replaced with the fix-ID. In the second line the values are replaced with the appropriate fields from the fix ave/time command. There is no third line in the header of the file, so the *title3* setting is ignored when *mode* = scalar.

By default, these header lines are as follows for *mode* = vector:

```
# Time-averaged data for fix ID
# TimeStep Number-of-rows
# Row value1 value2 ...
```

In the first line, ID is replaced with the fix-ID. The second line describes the two values that are printed at the first of each section of output. In the third line the values are replaced with the appropriate fields from the fix ave/time command.

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

Added in version 4May2022.

No information about this fix is written to *binary restart files*. The *fix_modify colname* option can be used to change the name of the column in the output file. When writing a YAML format file this name will be in the list of keywords.

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 time steps that are multiples of N_{freq} 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 produces a scalar or vector, then the scalar and each element of the vector can be either “intensive” or “extensive”, depending on whether the values contributing to the scalar or vector element are “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 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*.

2.23.5 Restrictions

none

2.23.6 Related commands

compute, *fix ave/atom*, *fix ave/chunk*, *fix ave/histo*, *variable*, *fix ave/correlate*,

2.23.7 Default

The option defaults are mode = scalar, ave = one, start = 0, no file output, format = %g, title 1,2,3 = strings as described above, and no off settings for any input values.

2.24 fix aveforce command

2.24.1 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

2.24.2 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
```

2.24.3 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*, , or *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 three quantities defining the force components, namely *fx*, *fy*, and *fz*, 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 average force.

Equal-style variables can specify formulas with various mathematical functions, and include *thermo_style* command keywords for the simulation box parameters and timestep and elapsed time. Thus it is easy to specify a time-dependent average force.

If the *region* keyword is used, the atom must also be in the specified geometric *region* in order to have force added to it.

Styles with a *gpu*, *intel*, *kk*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed on the [Accelerator packages](#) page. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the GPU, INTEL, KOKKOS, OPENMP, and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the [Build package](#) page for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the [-suffix command-line switch](#) when you invoke LAMMPS, or you can use the [suffix](#) command in your input script.

See the [Accelerator packages](#) page for more instructions on how to use the accelerated styles effectively.

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

No information about this fix is written to *binary restart files*.

The *fix_modify respa* option is supported by this fix. This allows to set at which level of the *r-RESPA* integrator the fix is adding its forces. Default is the outermost level.

This fix computes a global three-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.

2.24.5 Restrictions

none

2.24.6 Related commands

fix setforce, *fix addforce*

2.24.7 Default

none

2.25 fix balance command

2.25.1 Syntax

```
fix ID group-ID balance Nfreq thresh style args keyword args ...
```

- ID, group-ID are documented in [fix](#) command
- balance = style name of this fix command
- Nfreq = perform dynamic load balancing every this many steps
- thresh = imbalance threshold that must be exceeded to perform a re-balance
- style = *shift* or *rcb* or *report*

shift args = dimstr Niter stopthresh
 dimstr = sequence of letters containing x or y or z, each not more than once
 Niter = # of times to iterate within each dimension of dimstr sequence
 stopthresh = stop balancing when this imbalance threshold is reached
 rcb args = none
 report args = none

- zero or more keyword/arg pairs may be appended
- keyword = *weight* or *out*

weight style args = use weighted particle counts for the balancing
 style = group or neigh or time or var or store
 group args = Ngroup group1 weight1 group2 weight2 ...
 Ngroup = number of groups with assigned weights
 group1, group2, ... = group IDs
 weight1, weight2, ... = corresponding weight factors
 neigh factor = compute weight based on number of neighbors
 factor = scaling factor (> 0)
 time factor = compute weight based on time spent computing
 factor = scaling factor (> 0)
 var name = take weight from atom-style variable
 name = name of the atom-style variable
 store name = store weight in custom atom property defined by [fix property/atom](#) command
 name = atom property name (without d_ prefix)
 sort arg = no or yes
 out arg = filename
 filename = write each processor's subdomain to a file, at each re-balancing

2.25.2 Examples

```
fix 2 all balance 1000 1.05 shift x 10 1.05
fix 2 all balance 100 0.9 shift xy 20 1.1 out tmp.balance
fix 2 all balance 100 0.9 shift xy 20 1.1 weight group 3 substrate 3.0 solvent 1.0 solute 0.8 out tmp.balance
fix 2 all balance 100 1.0 shift x 10 1.1 weight time 0.8
fix 2 all balance 100 1.0 shift xy 5 1.1 weight var myweight weight neigh 0.6 weight store allweight
fix 2 all balance 1000 1.1 rcb
```

2.25.3 Description

This command adjusts the size and shape of processor subdomains within the simulation box, to attempt to balance the number of particles and thus the computational cost (load) evenly across processors. The load balancing is “dynamic” in the sense that re-balancing is performed periodically during the simulation. To perform “static” balancing, before or between runs, see the [balance](#) command.

Added in version 17Apr2024.

The *report* balance style only computes the load imbalance but does not attempt any re-balancing. This way the load imbalance information can be used otherwise, for instance for stopping a run with [fix halt](#).

Load-balancing is typically most useful if the particles in the simulation box have a spatially-varying density distribution or where the computational cost varies significantly between different atoms (e.g., a model of a vapor/liquid interface, or a solid with an irregular-shaped geometry containing void regions, or [hybrid pair style simulations](#) that combine pair styles with different computational cost). In these cases, the LAMMPS default of dividing the simulation box volume into a regular-spaced grid of 3d bricks, with one equal-volume subdomain per processor, may assign numbers of particles per processor in a way that the computational effort varies significantly. This can lead to poor performance when the simulation is run in parallel.

The balancing can be performed with or without per-particle weighting. With no weighting, the balancing attempts to assign an equal number of particles to each processor. With weighting, the balancing attempts to assign an equal aggregate computational weight to each processor, which typically induces a different number of atoms assigned to each processor.

Note

The weighting options listed above are documented with the [balance](#) command in [this section of the balance command](#) doc page. That section describes the various weighting options and gives a few examples of how they can be used. The weighting options are the same for both the fix balance and [balance](#) commands.

Note that the [processors](#) command allows some control over how the box volume is split across processors. Specifically, for a $P_x \times P_y \times P_z$ grid of processors, it allows choices of P_x , P_y , and P_z subject to the constraint that $P_x P_y P_z = P$, the total number of processors. This is sufficient to achieve good load-balance for some problems on some processor counts. However, all the processor subdomains will still have the same shape and the same volume.

On a particular time step, a load-balancing operation is only performed if the current “imbalance factor” in particles owned by each processor exceeds the specified *thresh* parameter. The imbalance factor is defined as the maximum number of particles (or weight) owned by any processor, divided by the average number of particles (or weight) per processor. Thus, an imbalance factor of 1.0 is perfect balance.

As an example, for 10000 particles running on 10 processors, if the most heavily loaded processor has 1200 particles, then the imbalance factor is 1.2, meaning there is a 20% imbalance. Note that re-balances can be forced even if the current balance is perfect (1.0) by specifying a *thresh* < 1.0.

Note

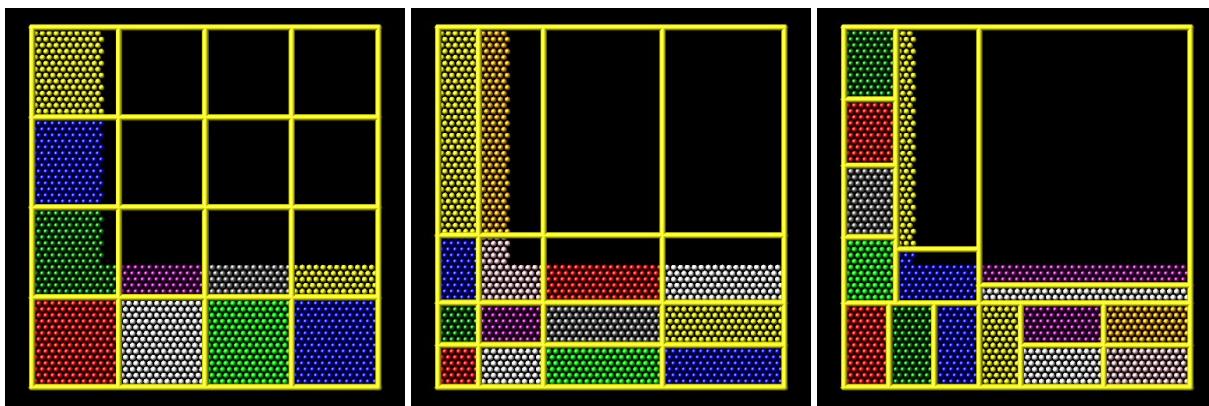
This command attempts to minimize the imbalance factor, as defined above. But depending on the method a perfect balance (1.0) may not be achieved. For example, “grid” methods (defined below) that create a logical 3d grid cannot achieve perfect balance for many irregular distributions of particles. Likewise, if a portion of the system is a perfect, non-rotated lattice (e.g., the initial system is generated by the [create_atoms](#) command with no rotations), then “grid” methods may be unable to achieve exact balance. This is because entire lattice planes will be owned or not owned by a single processor.

Note

The imbalance factor is also an estimate of the maximum speed-up you can hope to achieve by running a perfectly balanced simulation versus an imbalanced one. In the example above, the 10000-particle simulation could run up to 20% faster if it were perfectly balanced, versus when imbalanced. However, computational cost is not strictly proportional to particle count, and changing the relative size and shape of processor subdomains may lead to additional computational and communication overheads (e.g., in the PPPM solver used via the [kspace_style](#) command). Thus, you should benchmark the run times of a simulation before and after balancing.

The method used to perform a load balance is specified by one of the listed styles, which are described in detail below. There are two kinds of styles.

The *shift* style is a “grid” method which produces a logical 3d grid of processors. It operates by changing the cutting planes (or lines) between processors in 3d (or 2d), to adjust the volume (area in 2d) assigned to each processor, as in the following 2d diagram where processor subdomains are shown and atoms are colored by the processor that owns them.



The leftmost diagram is the default partitioning of the simulation box across processors (one sub-box for each of 16 processors); the middle diagram is after a “grid” method has been applied. The *rcb* style is a “tiling” method which does not produce a logical 3d grid of processors. Rather it tiles the simulation domain with rectangular sub-boxes of varying size and shape so as to have equal numbers of particles (or weight) in each sub-box, as in the rightmost diagram above.

The “grid” methods can be used with either of the [comm_style](#) command options, *brick* or *tiled*. The “tiling” methods can only be used with [comm_style tiled](#).

When a “grid” method is specified, the current domain partitioning can be either a logical 3d grid or a tiled partitioning. In the former case, the current logical 3d grid is used as a starting point and changes are made to improve the imbalance factor. In the latter case, the tiled partitioning is discarded and a logical 3d grid is created with uniform spacing in all dimensions. This is the starting point for the balancing operation.

When a “tiling” method is specified, the current domain partitioning (“grid” or “tiled”) is ignored, and a new partitioning is computed from scratch.

The *group-ID* is ignored. However the impact of balancing on different groups of atoms can be affected by using the *group* weight style as described below.

The *N_{freq}* setting determines how often a re-balance is performed. If *N_{freq}* > 0, then re-balancing will occur every *N_{freq}* steps. Each time a re-balance occurs, a renighboring is triggered, so *N_{freq}* should not be too small. If *N_{freq}* = 0, then re-balancing will be done every time renighboring normally occurs, as determined by the the [neighbor](#) and [neigh_modify](#) command settings.

On re-balance steps, re-balancing will only be attempted if the current imbalance factor, as defined above, exceeds the *thresh* setting.

The *shift* style invokes a “grid” method for balancing, as described above. It changes the positions of cutting planes between processors in an iterative fashion, seeking to reduce the imbalance factor.

The *dimstr* argument is a string of characters, each of which must be *x* or *y* or *z*. Each character can appear zero or one time, since there is no advantage to balancing on a dimension more than once. You should normally only list dimensions where you expect there to be a density variation in the particles.

Balancing proceeds by adjusting the cutting planes in each of the dimensions listed in *dimstr*, one dimension at a time. For a single dimension, the balancing operation (described below) is iterated on up to N_{iter} times. After each dimension finishes, the imbalance factor is re-computed, and the balancing operation halts if the *stopthresh* criterion is met.

A re-balance operation in a single dimension is performed using a density-dependent recursive multisectioning algorithm, where the position of each cutting plane (line in 2d) in the dimension is adjusted independently. This is similar to a recursive bisectioning for a single value, except that the bounds used for each bisectioning take advantage of information from neighboring cuts if possible, as well as counts of particles at the bounds on either side of each cuts, which themselves were cuts in previous iterations. The latter is used to infer a density of particles near each of the current cuts. At each iteration, the count of particles on either side of each plane is tallied. If the counts do not match the target value for the plane, the position of the cut is adjusted based on the local density. The low and high bounds are adjusted on each iteration, using new count information, so that they become closer together over time. Thus as the recursion progresses, the count of particles on either side of the plane gets closer to the target value.

The density-dependent part of this algorithm is often an advantage when you re-balance a system that is already nearly balanced. It typically converges more quickly than the geometric bisectioning algorithm used by the *balance* command. However, if can be a disadvantage if you attempt to re-balance a system that is far from balanced, and converge more slowly. In this case you probably want to use the *balance* command before starting a run, so that you begin the run with a balanced system.

Once the re-balancing is complete and final processor subdomains assigned, particles migrate to their new owning processor as part of the normal renighboring procedure.

Note

At each re-balance operation, the bisectioning for each cutting plane (line in 2d) typically starts with low and high bounds separated by the extent of a processor’s subdomain in one dimension. The size of this bracketing region shrinks based on the local density, as described above, which should typically be 1/2 or more every iteration. Thus if N_{iter} is specified as 10, the cutting plane will typically be positioned to better than 1 part in 1000 accuracy (relative to the perfect target position). For $N_{\text{iter}} = 20$, it will be accurate to better than 1 part in a million. Thus there is no need to set N_{iter} to a large value. This is especially true if you are re-balancing often enough that each time you expect only an incremental adjustment in the cutting planes is necessary. LAMMPS will check if the threshold accuracy is reached (in a dimension) is less iterations than N_{iter} and exit early.

The *rcb* style invokes a “tiled” method for balancing, as described above. It performs a recursive coordinate bisectioning (RCB) of the simulation domain. The basic idea is as follows.

The simulation domain is cut into two boxes by an axis-aligned cut in the longest dimension, leaving one new box on either side of the cut. All the processors are also partitioned into two groups, half assigned to the box on the lower side of the cut, and half to the box on the upper side. If the processor count is odd, one side gets an extra processor. The cut is positioned so that the number of atoms in the lower box is exactly the number that the processors assigned to that box should own for load balance to be perfect. This also makes load balance for the upper box perfect. The positioning is done iteratively, by a bisectioning method. Note that counting atoms on either side of the cut requires communication between all processors at each iteration.

That is the procedure for the first cut. Subsequent cuts are made recursively, in exactly the same manner. The subset of processors assigned to each box make a new cut in the longest dimension of that box, splitting the box, the subset of processors, and the atoms in the box in two. The recursion continues until every processor is assigned a sub-box of the entire simulation domain, and owns the atoms in that sub-box.

The *sort* keyword determines whether the communication of per-atom data to other processors during load-balancing will be random or deterministic. Random is generally faster; deterministic will ensure the new ordering of atoms on each processor is the same each time the same simulation is run. This can be useful for debugging purposes. Since the fix balance command is performed during timestepping, the default is *no* so that sorting is not performed.

The *out* keyword writes text to the specified *filename* with the results of each re-balancing operation. The file contains the bounds of the subdomain for each processor after the balancing operation completes. The format of the file is compatible with the [Pizza.py mdump](#) tool which has support for manipulating and visualizing mesh files. An example is shown here for a balancing by four processors for a 2d problem:

```
ITEM: Timestep
0
ITEM: Number of Nodes
16
ITEM: Box Bounds
0 10
0 10
0 10
ITEM: Nodes
1 1 0 0 0
2 1 5 0 0
3 1 5 5 0
4 1 0 5 0
5 1 5 0 0
6 1 10 0 0
7 1 10 5 0
8 1 5 5 0
9 1 0 5 0
10 1 5 5 0
11 1 5 10 0
12 1 10 5 0
13 1 5 5 0
14 1 10 5 0
15 1 10 10 0
16 1 5 10 0
ITEM: Timestep
0
ITEM: Number of Squares
4
ITEM: Squares
1 1 1 2 3 4
2 1 5 6 7 8
3 1 9 10 11 12
4 1 13 14 15 16
```

The coordinates of all the vertices are listed in the NODES section, five per processor. Note that the four subdomains share vertices, so there will be duplicate nodes in the list.

The “SQUARES” section lists the node IDs of the four vertices in a rectangle for each processor (1 to 4).

For a 3d problem, the syntax is similar but with eight vertices listed for each processor instead of four, and “SQUARES” replaced by “CUBES”.

2.25.4 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 is the imbalance factor after the most recent re-balance and a global vector of length 3 with additional information about the most recent re-balancing. The three values in the vector are as follows:

1. max # of particles per processor
2. total # iterations performed in last re-balance
3. imbalance factor right before the last re-balance was performed

As explained above, the imbalance factor is the ratio of the maximum number of particles (or total weight) on any processor to the average number of particles (or total weight) per processor.

These quantities can be accessed by various *output commands*. The scalar and vector values calculated by this fix are “intensive”.

No parameter of this fix can be used with the *start/stop* keywords of the *run* command. This fix is not invoked during *energy minimization*.

2.25.5 Restrictions

For 2d simulations, the *z* style cannot be used, nor can *z* appear in *dimstr* for the *shift* style.

Balancing through recursive bisectioning (*rcb* style) requires *comm_style tiled*.

2.25.6 Related commands

group, processors, balance, comm_style

2.25.7 Default

The default setting is sort = no.

2.26 fix bocs command

2.26.1 Syntax

```
fix ID group-ID bocs keyword values ...
```

- ID, group-ID are documented in *fix* command
- bocs = style name of this fix command
- two or more keyword/value pairs may be appended

- keyword = *temp* or *cgiso* or *tchain* or *pchain* or *mtk* or *tloop* or *ploop*

temp values = Tstart Tstop Tdamp
 cgiso values = Pstart Pstop Pdamp basis_set args
 basis_set = analytic or linear_spline or cubic_spline
 analytic args = V_avg N_particles N_coeff Coeff_1 Coeff_2 ... Coeff_N
 linear_spline args = input_filename
 cubic_spline args = input_filename
 tchain value = N = length of thermostat chain (1 = single thermostat)
 pchain value = N = length of thermostat on barostat (0 = no thermostat)
 mtk value = yes or no = add MTK adjustment term or not
 tloop value = M = number of sub-cycles to perform on thermostat
 ploop value = M = number of sub-cycles to perform on barostat

2.26.2 Examples

```
fix 1 all bocs temp 300.0 300.0 100.0 cgiso 0.986 0.986 1000.0 analytic 66476.015 968 2 245030.10 8962.20
fix 1 all bocs temp 300.0 300.0 100.0 cgiso 0.986 0.986 1000.0 cubic_spline input_Fv.dat
thermo_modify press 1_press
```

2.26.3 Description

These commands incorporate a pressure correction as described by Dunn and Noid ([Dunn1](#)) to the standard MTK barostat by Martyna et al. ([Martyna](#)). The first half of the command mimics a standard *fix npt* command:

```
fix 1 all bocs temp Tstart Tstop Tcoupl cgiso Pstart Pstop Pdamp
```

The two differences are replacing *npt* with *bocs*, and replacing *isot/aniso/etc.* with *cgiso*. The rest of the command details what form you would like to use for the pressure correction equation. The choices are: *analytic*, *linear_spline*, or *cubic_spline*.

With either spline method, the only argument that needs to follow it is the name of a file that contains the desired pressure correction as a function of volume. The file must be formatted so each line has:

```
Volume_i, PressureCorrection_i
```

Note both the COMMA and the SPACE separating the volume's value and its corresponding pressure correction. The volumes in the file must be uniformly spaced. Both the volumes and the pressure corrections should be provided in the proper units (e.g., if you are using *units real*, the volumes should all be in Å³ and the pressure corrections should all be in atm). Furthermore, the table should start/end at a volume considerably smaller/larger than you expect your system to sample during the simulation. If the system ever reaches a volume outside of the range provided, the simulation will stop.

With the *analytic* option, the arguments are as follows:

```
... analytic V_avg N_particles N_coeff Coeff_1 Coeff_2 ... Coeff_N
```

Note that *V_avg* and *Coeff_i* should all be in the proper units (e.g., if you are using *units real*, *V_avg* should be in Å³ and the coefficients should all be in atm · Å³).

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

This fix writes the cumulative global energy change to [binary restart files](#). See the [read_restart](#) command for info on how to re-specify a fix in an input script that reads a restart file, so that the fix continues in an uninterrupted fashion.

The [fix_modify temp](#) option is supported by this fix. You can use it to assign a temperature [compute](#) you have defined to this fix which will be used in its thermostating procedure, as described above. For consistency, the group used by this fix and by the compute should be the same.

The cumulative energy change in the system imposed by this fix is included in the [thermodynamic output](#) keywords [ecouple](#) and [econserve](#). See the [thermo_style](#) doc page for details.

This fix computes a global scalar which can be accessed by various [output commands](#). The scalar is the same cumulative energy change due to this fix described in the previous paragraph. The scalar value calculated by this fix is “extensive”.

This fix can ramp its target temperature over multiple runs, using the [start](#) and [stop](#) keywords of the [run](#) command. See the [run](#) command for details of how to do this.

This fix is not invoked during [energy minimization](#).

2.26.5 Restrictions

This fix is part of the BOCS package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

As this is computing a (modified) pressure, group-ID should be *all*.

The pressure correction has only been tested for use with an isotropic pressure coupling in 3 dimensions.

By default, LAMMPS will still report the normal value for the pressure if the pressure is printed via a [thermo](#) command, or if the pressures are written to a file every so often. In order to have LAMMPS report the modified pressure, you must include the [thermo_modify](#) command given in the examples. For the last argument in the command, you should put XXXX_press, where XXXX is the ID given to the fix bocs command (in the example, the ID of the fix bocs command is 1).

2.26.6 Further information

For more details about the pressure correction and the entire BOCS software package, visit the BOCS package on [GitHub](#) and read the release paper by Dunn et al. ([Dunn2](#)) .

(**Dunn1**) Dunn and Noid, J Chem Phys, 143, 243148 (2015).

(**Martyna**) Martyna, Tobias, and Klein, J Chem Phys, 101, 4177 (1994).

(**Dunn2**) Dunn, Lebold, DeLyser, Rudzinski, and Noid, J. Phys. Chem. B, 122, 3363 (2018).

2.27 fix bond/break command

2.27.1 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 (integer or type label)
- Rmax = bond longer than Rmax can break (distance units)
- zero or more keyword/value pairs may be appended
- keyword = *prob*
 prob values = fraction seed
 fraction = break a bond with this probability if otherwise eligible
 seed = random number seed (positive integer)

2.27.2 Examples

```
fix 5 all bond/break 10 2 1.2
fix 5 polymer bond/break 1 1 2.0 prob 0.5 49829
```

2.27.3 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, as will all angle, dihedral, and improper interactions that bond is part of.

This is different than a [pair-wise](#) bond-order potential such as Tersoff or AIROBO which infers bonds and many-body interactions based on the current geometry of a small cluster of atoms and effectively creates and destroys bonds and higher-order many-body interactions from timestep to timestep as atoms move.

A check for possible bond breakage is performed every *Nevery* timesteps. If two bonded atoms *i* and *j* are farther than the distance *Rmax* from each other, the bond is of type *bondtype*, and both *i* and *j* are in the specified fix group, then the bond between *i* and *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 *j* in its sole bond, then the bond between *i* and *j* 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 time step. It also means that if atom *i* chooses atom *j* as its sole partner, but atom *j* chooses atom *k* as its sole partner (because $R_{jk} > R_{ij}$), then this means atom *i* will not be part of a broken bond on this time step, 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 is less than *fraction*.

When a bond is broken, data structures within LAMMPS that store bond topologies are updated to reflect the breakage. Likewise, if the bond is part of a 3-body (angle) or 4-body (dihedral, improper) interaction, that interaction is removed as well. These changes typically affect pair-wise interactions between atoms that used to be part of bonds, angles, etc.

Note

One data structure that is not updated when a bond breaks are the molecule IDs stored by each atom. Even though one molecule becomes two molecules due to the broken bond, all atoms in both new molecules retain their original molecule IDs.

Computationally, each time step this fix is invoked, it loops over all the bonds in the system and computes distances between pairs of bonded atoms. It also communicates between neighboring processors to coordinate which bonds are broken. Moreover, if any bonds are broken, neighbor lists must be immediately updated on the same time step. This is to ensure that any pair-wise interactions that should be turned “on” due to a bond breaking, because they are no longer excluded by the presence of the bond and the settings of the [special_bonds](#) command, will be immediately recognized. All of these operations increase the cost of a time step. 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.

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 two atoms are separated by a distance far from the equilibrium bond length, then the two 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 (as well as angles, dihedrals, and impropers).

See the [Howto](#) page on broken bonds for more information on related features in LAMMPS.

2.27.4 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. This vector can be accessed by various [output commands](#). The vector values calculated by this fix are “intensive”.

The two quantities in the global vector are

- (1) number of bonds broken on the most recent breakage time step
- (2) cumulative number 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](#).

2.27.5 Restrictions

This fix is part of the MC package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) doc page for more info.

2.27.6 Related commands

fix bond/create, fix bond/react, fix bond/swap, dump local, special_bonds

2.27.7 Default

The option defaults are prob = 1.0.

2.28 fix bond/create command

2.29 fix bond/create/angle command

2.29.1 Syntax

```
fix ID group-ID style Nevery itype jtype Rmin bondtype keyword values ...
```

- ID, group-ID are documented in [fix](#) command
 - style = *bond/create* or *bond/create/angle*
 - Nevery = attempt bond creation every this many steps
 - itype,jtype = atoms of itype can bond to atoms of jtype (1-Ntypes or type label)
 - Rmin = two atoms separated by less than Rmin can bond (distance units)
 - bondtype = type of created bonds (integer or type label)
 - zero or more keyword/value pairs may be appended to args
 - keyword = *iparam* or *jparam* or *prob* or *atype* or *dtype* or *itype* or *aconstraint*
- 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 (1-Ntypes or type label)
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 (1-Ntypes or type label)
prob values = fraction seed
fraction = create a bond with this probability if otherwise eligible
seed = random number seed (positive integer)
atype value = angletype
angletype = type of created angles (integer or type label)
dtype value = dihedraltype
dihedraltype = type of created dihedrals (integer or type label)
itype value = impropertype
impropertype = type of created impropers (integer or type label)
aconstraint value = amin amax

amin = minimal angle at which new bonds can be created
amax = maximal angle at which new bonds can be created

2.29.2 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
fix 5 all bond/create 1 3 3 0.8 1 prob 0.5 85784 iparam 2 3 atype 1 dtype 2
fix 5 all bond/create/angle 10 1 2 1.122 1 aconstrain 120 180 prob 1 4928459 iparam 2 1 jparam 2 2

labelmap atom 1 c1 2 n2
labelmap bond 1 c1-n2
fix 5 all bond/create 10 c1 n2 0.8 c1-n2
```

2.29.3 Description

Create bonds between pairs of atoms as a simulation runs according to specified criteria. This can be used to model the 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. Optionally, the creation of a bond can also create angle, dihedral, and improper interactions that the bond is part of. See the discussion of the *atype*, *dtype*, and *itype* keywords below.

This process is different than a [pair-wise](#) bond-order potential such as Tersoff or AIREBO, which infer bonds and many-body interactions based on the current geometry of a small cluster of atoms and effectively create and destroy bonds and higher-order many-body interactions from time step to time step as the atoms move.

A check for possible new bonds is performed every *Nevery* time steps. If two atoms *i* and *j* are within a distance *Rmin* of each other, atom *i* is of type *itype*, atom *j* is of type *jtype*, and both *i* and *j* are in the specified fix group, then if a bond does not already exist between atoms *i* and *j*, and if both *i* and *j* meet their respective *maxbond* requirements (explained below), then *i* and *j* are 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 that an atom will only be part of at most one created bond on a given time step. It also means that if atom *i* chooses atom *j* as its sole partner, but atom *j* chooses atom *k* as its sole partner (because $R_{jk} < R_{ij}$), then atom *i* will not form a bond on this time step, even if it has other possible bond partners.

It is permissible to have *itype* = *jtype*. *Rmin* must be \leq the pair-wise 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 type *itype* already has *maxbond* bonds (as set by the *iparam* keyword), then it will not form any more, and 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 pair-wise 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 affect 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 is less than *fraction*.

The *aconstraint* keyword is only available with the fix bond/create/angle command. It allows one to specify minimum and maximum angles *amin* and *amax*, respectively, between the two prospective bonding partners and a third particle that is already bonded to one of the two partners. Such a criterion can be important when new angles are defined together with the formation of a new bond. Without a restriction on the permissible angle, and for stiffer angle potentials, very large energies can arise and lead to unphysical behavior.

Any bond that is created is assigned a bond type of *bondtype*.

When a bond is created, data structures within LAMMPS that store bond topologies are updated to reflect the creation. If the bond is part of new 3-body (angle) or 4-body (dihedral, improper) interactions, you can choose to create new angles, dihedrals, and impropers as well using the *atype*, *dtype*, and *itype* keywords. All of these changes typically affect pair-wise interactions between atoms that are now part of new bonds, angles, etc.

Note

One data structure that is not updated when a bond breaks are the molecule IDs stored by each atom. Even though two molecules become one molecule due to the created bond, all atoms in the new molecule retain their original molecule IDs.

If the *atype* keyword is used and if an angle potential is defined via the [angle_style](#) command, then any new 3-body interactions inferred by the creation of a bond will create new angles of type *angletype*, with parameters assigned by the corresponding [angle_coeff](#) command. Likewise, the *dtype* and *itype* keywords will create new dihedrals and impropers of type *dihedraltypes* and *impropertype*.

Note

To create a new bond, the internal LAMMPS data structures that store this information must have space for it. When LAMMPS is initialized from a data file, the list of bonds is scanned and the maximum number of bonds per atom is tallied. If some atom will acquire more bonds than this limit as this fix operates, then the “extra bond per atom” parameter must be set to allow for it. Ditto for “extra angle per atom”, “extra dihedral per atom”, and “extra improper per atom” if angles, dihedrals, or impropers are being added when bonds are created. See the [read_data](#) or [create_box](#) command for more details. Note that a data file with no atoms can be used if you wish to add non-bonded atoms via the [create atoms](#) command (e.g., for a percolation simulation).

Note

LAMMPS stores and maintains a data structure with a list of the first, second, and third neighbors of each atom (within the bond topology of the system) for use in weighting pair-wise interactions for bonded atoms. Note that adding a single bond always adds a new first neighbor but may also induce **many** new second and third neighbors, depending on the molecular topology of your system. The “extra special per atom” parameter must typically be set to allow for the new maximum total size (first + second + third neighbors) of this per-atom list. There are two ways to do this. See the [read_data](#) or [create_box](#) commands for details.

Note

Even if you do not use the *atype*, *dtype*, or *itype* keywords, the list of topological neighbors is updated for atoms affected by the new bond. This in turn affects which neighbors are considered for pair-wise interactions, using the weighting rules set by the [special_bonds](#) command. Consider a new bond created between atoms *i* and *j*. If *j* has a bonded neighbor *k*, then *k* becomes a second neighbor of *i*. Even if the *atype* keyword is not used to create angle $\angle ijk$, the pair-wise interaction between *i* and *k* could potentially be turned off or weighted by the 1-3

weighting specified by the `special_bonds` command. This is the case even if the “angle yes” option was used with that command. The same is true for third neighbors (1–4 interactions), the `dtype` keyword, and the “dihedral yes” option used with the `special_bonds` command.

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 pair-wise coefficients must be specified for the new types.

Computationally, each time step this fix is invoked, 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. Moreover, if any bonds are created, neighbor lists must be immediately updated on the same time step. This is to ensure that any pair-wise interactions that should be turned “off” due to a bond creation, because they are now excluded by the presence of the bond and the settings of the `special_bonds` command, will be immediately recognized. All of these operations increase the cost of a time step. 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.

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 two atoms are separated by a distance far from the equilibrium bond length, then the two 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 (and angles, dihedrals, impropers).

2.29.4 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”.

The two quantities in the global vector are

- (1) number of bonds created on the most recent creation time step
- (2) cumulative number 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`.

2.29.5 Restrictions

This fix is part of the MC package. It is only enabled if LAMMPS was built with that package. See the `Build package` doc page for more info.

2.29.6 Related commands

fix bond/break, fix bond/react, fix bond/swap, dump local, special_bonds

2.29.7 Default

The option defaults are iparam = (0,itype), jparam = (0,jtype), and prob = 1.0.

2.30 fix bond/react command

2.30.1 Syntax

```
fix ID group-ID bond/react common_keyword values &
  react react-ID react-group-ID Nevery Rmin Rmax template-ID(pre-reacted) template-ID(post-reacted) &
  ↵map_file individual_keyword values &
  react react-ID react-group-ID Nevery Rmin Rmax template-ID(pre-reacted) template-ID(post-reacted) &
  ↵map_file individual_keyword values &
  react react-ID react-group-ID Nevery Rmin Rmax template-ID(pre-reacted) template-ID(post-reacted) &
  ↵map_file individual_keyword values &
  ...
  ...
```

- ID, group-ID are documented in [fix](#) command.
- bond/react = style name of this fix command
- the common keyword/values may be appended directly after ‘bond/react’
- common keywords apply to all reaction specifications
- common_keyword = *stabilization* or *reset_mol_ids*
 - stabilization values = stabilize group_prefix xmax
 - stabilize = yes or no
 - yes = perform reaction site stabilization
 - no = no reaction site stabilization (default)
 - group_prefix = user-assigned prefix for the dynamic group of atoms not currently involved in a reaction
 - xmax = value that is used by an internally-created [nve/limit](#) integrator
 - reset_mol_ids values = yes or no
 - yes = update molecule IDs based on new global topology (default)
 - no = do not update molecule IDs
- react = mandatory argument indicating new reaction specification
- react-ID = user-assigned name for the reaction
- react-group-ID = only atoms in this group are considered for the reaction
- Nevery = attempt reaction every this many steps
- Rmin = initiator atoms must be separated by more than Rmin to initiate reaction (distance units)
- Rmax = initiator atoms must be separated by less than Rmax to initiate reaction (distance units)
- template-ID(pre-reacted) = ID of a molecule template containing pre-reaction topology
- template-ID(post-reacted) = ID of a molecule template containing post-reaction topology

- map_file = name of file specifying corresponding atom-IDs in the pre- and post-reacted templates
- zero or more individual keyword/value pairs may be appended to each react argument
- individual_keyword = prob or rate_limit or max_rxn or stabilize_steps or custom_charges or rescale_charges or molecule or modify_create

prob values = fraction seed

fraction = initiate reaction with this probability if otherwise eligible

seed = random number seed (positive integer)

rate_limit = Nlimit Nsteps

Nlimit = maximum number of reactions allowed to occur within interval

Nsteps = the interval (number of timesteps) over which to count reactions

max_rxn value = N

N = maximum number of reactions allowed to occur

stabilize_steps value = timesteps

timesteps = number of time steps to apply the internally-created nve/limit fix to reacting atoms

custom_charges value = no or fragment-ID

no = update all atomic charges (default)

fragment-ID = ID of molecule fragment whose charges are updated

rescale_charges value = no or yes

no = do not rescale atomic charges (default)

yes = rescale charges such that total charge does not change during reaction

molecule value = off or inter or intra

off = allow both inter- and intramolecular reactions (default)

inter = search for reactions between molecules with different IDs

intra = search for reactions within the same molecule

modify_create values = keyword arg

fit arg = all or fragment-ID

all = use all eligible atoms for create-atoms fit (default)

fragment-ID = ID of molecule fragment used for create-atoms fit

overlap value = R

R = only insert atom/molecule if further than R from existing particles (distance units)

2.30.2 Examples

For unabridged example scripts and files, see examples/PACKAGES/reaction.

```

molecule mol1 pre_reacted_topology.txt
molecule mol2 post_reacted_topology.txt
fix 5 all bond/react react myrxn1 all 1 0 3.25 mol1 mol2 map_file.txt

molecule mol1 pre_reacted_rxn1.txt
molecule mol2 post_reacted_rxn1.txt
molecule mol3 pre_reacted_rxn2.txt
molecule mol4 post_reacted_rxn2.txt
fix 5 all bond/react stabilization yes nvt_grp .03 &
  react myrxn1 all 1 0 3.25 mol1 mol2 map_file_rxn1.txt prob 0.50 12345 &
  react myrxn2 all 1 0 2.75 mol3 mol4 map_file_rxn2.txt prob 0.25 12345
fix 6 nvt_grp.REACT nvt temp 300 300 100 # set thermostat after bond/react

```

2.30.3 Description

Initiate complex covalent bonding (topology) changes. These topology changes will be referred to as ‘reactions’ throughout this documentation. Topology changes are defined in pre- and post-reaction molecule templates and can include creation and deletion of bonds, angles, dihedrals, impropers, bond types, angle types, dihedral types, atom types, or atomic charges. In addition, reaction by-products or other molecules can be identified and deleted. Finally, atoms can be created and inserted at specific positions relative to the reaction site.

Fix bond/react does not use quantum mechanical (e.g., [fix qmmm](#)) or pairwise bond-order potential (e.g., [Tersoff](#) or [AIREBO](#)) methods to determine bonding changes a priori. Rather, it uses a distance-based probabilistic criteria to effect predetermined topology changes in simulations using standard force fields.

This fix was created to facilitate the dynamic creation of polymeric, amorphous or highly cross-linked systems. A suggested workflow for using this fix is

- (1) identify a reaction to be simulated
- (2) build a molecule template of the reaction site before the reaction has occurred
- (3) build a molecule template of the reaction site after the reaction has occurred
- (4) create a map that relates the template-atom-IDs of each atom between pre- and post-reaction molecule templates
- (5) fill a simulation box with molecules and run a simulation with fix bond/react.

Note

Added in version 15Sep2022.

Type labels allow for molecule templates and data files to use alphanumeric atom types that match those of a force field. Input files that use type labels are inherently compatible with each other and portable between different simulations. Therefore, it is highly recommended to use type labels to specify atom, bond, etc. types when using fix bond/react.

Only one ‘fix bond/react’ command can be used at a time. Multiple reactions can be simultaneously applied by specifying multiple *react* arguments to a single ‘fix bond/react’ command. This syntax is necessary because the “common” keywords are applied to all reactions.

The *stabilization* keyword enables reaction site stabilization. Reaction site stabilization is performed by including reacting atoms in an internally-created fix [nve/limit](#) time integrator for a set number of time steps given by the *stabilize_steps* keyword. While reacting atoms are being time integrated by the internal nve/limit, they are prevented from being involved in any new reactions. The *xmax* value keyword should typically be set to the maximum distance that non-reacting atoms move during the simulation.

Fix bond/react creates and maintains two important dynamic groups of atoms when using the *stabilization* keyword. The first group contains all atoms currently involved in a reaction; this group is automatically time-integrated by an internally-created [nve/limit](#) integrator. The second group contains all atoms currently not involved in a reaction. This group should be controlled by a thermostat in order to time integrate the system. The name of this group of non-reacting atoms is created by appending ‘_REACT’ to the group-ID argument of the *stabilization* keyword, as shown in the second example above.

Note

When using reaction stabilization, you should generally **not** have a separate thermostat that acts on the “all” group.

The group-ID set using the *stabilization* keyword can be an existing static group or a previously-unused group-ID. It cannot be specified as “all”. If the group-ID is previously unused, the fix bond/react command creates a *dynamic*

group that is initialized to include all atoms. If the group-ID is that of an existing static group, the group is used as the parent group of new, internally-created dynamic group. In both cases, this new dynamic group is named by appending '_REACT' to the group-ID (e.g., nvt_grp.REACT). By specifying an existing group, you may thermostat constant-topology parts of your system separately. The dynamic group contains only atoms not involved in a reaction at a given time step, and therefore should be used by a subsequent system-wide time integrator such as *fix nvt*, *fix npt*, or *fix nve*, as shown in the second example above (full examples can be found in examples/PACKAGES/reaction). The time integration command should be placed after the fix bond/react command due to the internal dynamic grouping performed by fix bond/react.

Note

If the group-ID is an existing static group, react-group-IDs should also be specified as this static group or a subset.

The *reset_mol_ids* keyword invokes the *reset_atoms mol* command after a reaction occurs, to ensure that molecule IDs are consistent with the new bond topology. The group-ID used for *reset_atoms mol* is the group-ID for this fix. Resetting molecule IDs is necessarily a global operation, so it can be slow for very large systems.

The following comments pertain to each *react* argument (in other words, they can be customized for each reaction, or reaction step):

A check for possible new reaction sites is performed every *Nevery* time steps. *Nevery* can be specified with an equal-style *variable*, whose value is rounded up to the nearest integer.

Three physical conditions must be met for a reaction to occur. First, an initiator atom pair must be identified within the reaction distance cutoffs. Second, the topology surrounding the initiator atom pair must match the topology of the pre-reaction template. Only atom types and bond connectivity are used to identify a valid reaction site (not bond types, etc.). Finally, any reaction constraints listed in the map file (see below) must be satisfied. If all of these conditions are met, the reaction site is eligible to be modified to match the post-reaction template.

An initiator atom pair will be identified if several conditions are met. First, a pair of atoms *i* and *j* within the specified react-group-ID of type *itype* and *jtype* must be separated by a distance between *Rmin* and *Rmax*. *Rmin* and *Rmax* can be specified with equal-style *variables*. For example, these reaction cutoffs can be functions of the reaction conversion using the following commands:

```
variable rmax equal 0 # initialize variable before bond/react
fix myrxn all bond/react react myrxn1 all 1 0 v_rmax mol1 mol2 map_file.txt
variable rmax equal 3+f_myrxn[1]/100 # arbitrary function of reaction count
```

The following criteria are used if multiple candidate initiator atom pairs are identified within the cutoff distance:

- (1) If the initiator atoms in the pre-reaction template are not 1–2 neighbors (i.e., not directly bonded) the closest potential partner is chosen.
- (2) Otherwise, if the initiator atoms in the pre-reaction template are 1–2 neighbors (i.e. directly bonded) the farthest potential partner is chosen.
- (3) Then, if both an atom *i* and atom *j* have each other as initiator partners, these two atoms are identified as the initiator atom pair of the reaction site.

Note that it can be helpful to select unique atom types for the initiator atoms: if an initiator atom pair is identified, as described in the previous steps, but it does not correspond to the same pair specified in the pre-reaction template, an otherwise eligible reaction could be prevented from occurring. Once this unique initiator atom pair is identified for each reaction, there could be two or more reactions that involve the same atom on the same time step. If this is the case, only one such reaction is permitted to occur. This reaction is chosen randomly from all potential reactions involving the overlapping atom. This capability allows, for example, different reaction pathways to proceed from identical reaction sites with user-specified probabilities.

The pre-reacted molecule template is specified by a molecule command. This molecule template file contains a sample reaction site and its surrounding topology. As described below, the initiator atom pairs of the pre-reacted template are specified by atom ID in the map file. The pre-reacted molecule template should contain as few atoms as possible while still completely describing the topology of all atoms affected by the reaction (which includes all atoms that change atom type or connectivity, and all bonds that change bond type). For example, if the force field contains dihedrals, the pre-reacted template should contain any atom within three bonds of reacting atoms.

Some atoms in the pre-reacted template that are not reacting may have missing topology with respect to the simulation. For example, the pre-reacted template may contain an atom that, in the simulation, is currently connected to the rest of a long polymer chain. These are referred to as edge atoms, and are also specified in the map file. All pre-reaction template atoms should be linked to an initiator atom, via at least one path that does not involve edge atoms. When the pre-reaction template contains edge atoms, not all atoms, bonds, charges, etc. specified in the reaction templates will be updated. Specifically, topology that involves only atoms that are “too near” to template edges will not be updated. The definition of “too near the edge” depends on which interactions are defined in the simulation. If the simulation has defined dihedrals, atoms within two bonds of edge atoms are considered “too near the edge.” If the simulation defines angles, but not dihedrals, atoms within one bond of edge atoms are considered “too near the edge.” If just bonds are defined, only edge atoms are considered “too near the edge.”

Note

Small molecules (i.e., ones that have all their atoms contained within the reaction templates) never have edge atoms.

Note that some care must be taken when building a molecule template for a given simulation. All atom types in the pre-reacted template must be the same as those of a potential reaction site in the simulation. A detailed discussion of matching molecule template atom types with the simulation is provided on the [molecule](#) command page. It is highly recommended to use [Type labels](#) (added in version 15Sep2022) in both molecule templates and data files, which automates the process of syncing atom types between different input files.

The post-reacted molecule template contains a sample of the reaction site and its surrounding topology after the reaction has occurred. It must contain the same number of atoms as the pre-reacted template (unless there are created atoms). A one-to-one correspondence between the atom IDs in the pre- and post-reacted templates is specified in the map file as described below. Note that during a reaction, an atom, bond, etc. type may change to one that was previously not present in the simulation. These new types must also be defined during the setup of a given simulation. A discussion of correctly handling this is also provided on the [molecule](#) command page.

Note

When a reaction occurs, it is possible that the resulting topology/atom (e.g., special bonds, dihedrals) exceeds that of the existing system and reaction templates. As when inserting molecules, enough space for this increased topology/atom must be reserved by using the relevant “extra” keywords to the [read_data](#) or [create_box](#) commands.

The map file is a text document with the following format:

A map file has a header and a body. The header of map file the contains one mandatory keyword and five optional keywords. The mandatory keyword is *equivalences*:

N equivalences = # of atoms N in the reaction molecule templates

The optional keywords are *edgeIDs*, *deleteIDs*, *chiralIDs*, and *constraints*:

N edgeIDs = # of edge atoms N in the pre-reacted molecule template

N deleteIDs = # of atoms N that are deleted

N createIDs = # of atoms N that are created

N chiralIDs = # of chiral centers N

N constraints = # of reaction constraints N

The body of the map file contains two mandatory sections and five optional sections. The first mandatory section begins with the keyword “InitiatorIDs” and lists the two atom IDs of the initiator atom pair in the pre-reacted molecule template. The second mandatory section begins with the keyword “Equivalences” and lists a one-to-one correspondence between atom IDs of the pre- and post-reacted templates. The first column is an atom ID of the pre-reacted molecule template, and the second column is the corresponding atom ID of the post-reacted molecule template. The first optional section begins with the keyword “EdgeIDs” and lists the atom IDs of edge atoms in the pre-reacted molecule template. The second optional section begins with the keyword “DeleteIDs” and lists the atom IDs of pre-reaction template atoms to delete. The third optional section begins with the keyword “CreateIDs” and lists the atom IDs of the post-reaction template atoms to create. The fourth optional section begins with the keyword “ChiralIDs” lists the atom IDs of chiral atoms whose handedness should be enforced. The fifth optional section begins with the keyword “Constraints” and lists additional criteria that must be satisfied in order for the reaction to occur. Currently, there are six types of constraints available, as discussed below: “distance”, “angle”, “dihedral”, “arrhenius”, “rmsd”, and “custom”.

A sample map file is given below:

```
# this is a map file
```

```
7 equivalences
2 edgeIDs
```

```
InitiatorIDs
```

```
3
5
```

```
EdgeIDs
```

```
1
7
```

```
Equivalences
```

```
1 1
2 2
3 3
4 4
5 5
6 6
7 7
```

A user-specified set of atoms can be deleted by listing their pre-reaction template IDs in the DeleteIDs section. A deleted atom must still be included in the post-reaction molecule template, in which it cannot be bonded to an atom that is not deleted. In addition to deleting unwanted reaction by-products, this feature can be used to remove specific topologies, such as small rings, that may be otherwise indistinguishable.

Atoms can be created by listing their post-reaction template IDs in the CreateIDs section. A created atom should not be included in the pre-reaction template. The inserted positions of created atoms are determined by the coordinates of the post-reaction template, after optimal translation and rotation of the post-reaction template to the reaction site (using a fit with atoms that are neither created nor deleted). The *modify_create* keyword can be used to modify the default behavior when creating atoms. The *modify_create* keyword has two sub-keywords, *fit* and *overlap*. One or more of the sub-keywords may be used after the *modify_create* keyword. The *fit* sub-keyword can be used to specify which post-reaction atoms are used for the optimal translation and rotation of the post-reaction template. The fragment-ID

value of the *fit* sub-keyword must be the name of a molecule fragment defined in the post-reaction *molecule* template, and only atoms in this fragment are used for the fit. Atoms are created only if no current atom in the simulation is within a distance R of any created atom, including the effect of periodic boundary conditions if applicable. R is defined by the *overlap* sub-keyword. Note that the default value for R is 0.0, which will allow atoms to strongly overlap if you are inserting where other atoms are present. The velocity of each created atom is initialized in a random direction with a magnitude calculated from the instantaneous temperature of the reaction site.

Note

The ‘Coords’ section must be included in the post-reaction template when creating atoms because these coordinates are used to determine where new atoms are inserted.

The handedness of atoms that are chiral centers can be enforced by listing their IDs in the ChiralIDs section. A chiral atom must be bonded to four atoms with mutually different atom types. This feature uses the coordinates and types of the involved atoms in the pre-reaction template to determine handedness. Three atoms bonded to the chiral center are arbitrarily chosen, to define an oriented plane, and the relative position of the fourth bonded atom determines the chiral center’s handedness.

Any number of additional constraints may be specified in the Constraints section of the map file. The constraint of type “distance” has syntax as follows:

distance ID1 ID2 rmin rmax

where “distance” is the required keyword, *ID1* and *ID2* are pre-reaction atom IDs (or molecule-fragment IDs, see below), and these two atoms must be separated by a distance between *rmin* and *rmax* for the reaction to occur.

The constraint of type “angle” has the following syntax:

angle ID1 ID2 ID3 amin amax

where “angle” is the required keyword, *ID1*, *ID2* and *ID3* are pre-reaction atom IDs (or molecule-fragment IDs, see below), and these three atoms must form an angle between *amin* and *amax* for the reaction to occur (where *ID2* is the central atom). Angles must be specified in degrees. This constraint can be used to enforce a certain orientation between reacting molecules.

The constraint of type “dihedral” has the following syntax:

dihedral ID1 ID2 ID3 ID4 amin amax amin2 amax2

where “dihedral” is the required keyword, and *ID1*, *ID2*, *ID3* and *ID4* are pre-reaction atom IDs (or molecule-fragment IDs, see below). Dihedral angles are calculated in the interval $(-180^\circ, 180^\circ]$. Refer to the *dihedral style* documentation for further details on convention. If *amin* is less than *amax*, these four atoms must form a dihedral angle greater than *amin* **and** less than *amax* for the reaction to occur. If *amin* is greater than *amax*, these four atoms must form a dihedral angle greater than *amin* **or** less than *amax* for the reaction to occur. Angles must be specified in degrees. Optionally, a second range of permissible angles *amin2* to *amax2* can be specified.

For the ‘distance’, ‘angle’, and ‘dihedral’ constraints (explained above), atom IDs can be replaced by pre-reaction molecule-fragment IDs. The molecule-fragment ID must begin with a letter. The location of the ID is the geometric center of all atom positions in the fragment. The molecule fragment must have been defined in the *molecule* command for the pre-reaction template.

The constraint of type ‘arrhenius’ imposes an additional reaction probability according to the modified Arrhenius equation,

$$k = AT^n e^{-E_a/k_B T}.$$

The Arrhenius constraint has the following syntax:

arrhenius A n E_a seed

where “arrhenius” is the required keyword, A is the pre-exponential factor, n is the exponent of the temperature dependence, E_a is the activation energy (*units* of energy), and *seed* is a random number seed. The temperature is defined as the instantaneous temperature averaged over all atoms in the reaction site and is calculated in the same manner as for example [compute temp/chunk](#). Currently, there are no options for additional temperature averaging or velocity-biased temperature calculations. A uniform random number between 0 and 1 is generated using *seed*; if this number is less than the result of the Arrhenius equation above, the reaction is permitted to occur.

The constraint of type ‘rmsd’ has the following syntax:

```
rmsd RMSDmax molfragment
```

where “rmsd” is the required keyword, and *RMSDmax* is the maximum root-mean-square deviation between atom positions of the pre-reaction template and the local reaction site (distance units), after optimal translation and rotation of the pre-reaction template. Optionally, the name of a molecule fragment (of the pre-reaction template) can be specified by *molfragment*. If a molecule fragment is specified, only atoms that are part of this molecule fragment are used to determine the RMSD. A molecule fragment must have been defined in the [molecule](#) command for the pre-reaction template. For example, the molecule fragment could consist of only the backbone atoms of a polymer chain. This constraint can be used to enforce a specific relative position and orientation between reacting molecules.

Changed in version 22Dec2022.

The constraint of type “custom” has the following syntax:

```
custom varstring
```

where ‘custom’ is the required keyword, and *varstring* is a variable expression. The expression must be a valid equal-style variable formula that can be read by the [variable](#) command, after any special reaction functions are evaluated. If the resulting expression is zero, the reaction is prevented from occurring; otherwise, it is permitted to occur. There are three special reaction functions available, ‘rxnbond’, ‘rxnsum’, and ‘rxnave’. The ‘rxnbond’ function allows per-bond values to be included in the variable strings of the custom constraint. The ‘rxnbond’ function has two mandatory arguments. The first argument is the ID of a previously defined ‘compute bond/local’ command. This ‘compute bond/local’ must compute only one value, e.g. bond force. This value is returned by the ‘rxnbond’ function. The second argument is the name of a molecule fragment in the pre-reaction template. The fragment must contain exactly two atoms, corresponding to the atoms involved in the bond whose value should be calculated. An example of a constraint that uses the force experienced by a bond is provided below. The ‘rxnsum’ and ‘rxnave’ functions operate over the atoms in a given reaction site, and have one mandatory argument and one optional argument. The mandatory argument is the identifier for an atom-style variable. The second, optional argument is the name of a molecule fragment in the pre-reaction template, and can be used to operate over a subset of atoms in the reaction site. The ‘rxnsum’ function sums the atom-style variable over the reaction site, while the ‘rxnave’ returns the average value. For example, a constraint on the total potential energy of atoms involved in the reaction can be imposed as follows:

```
compute 1 all pe/atom # in LAMMPS input script
variable my_pe atom c_1 # in LAMMPS input script
```

```
custom "rxnsum(v_my_pe) > 100" # in Constraints section of map file
```

The above example prevents the reaction from occurring unless the total potential energy of the reaction site is above 100. As a second example, this time using the ‘rxnbond’ function, consider a modified Arrhenius constraint that depends on the bond force of a specific bond:

```
# in LAMMPS input script

compute bondforce all bond/local force

compute ke_atom all ke/atom
variable ke atom c_ke_atom
```

(continues on next page)

(continued from previous page)

```
variable E_a equal 100.0 # activation energy
variable l0 equal 1.0 # characteristic length
```

in Constraints section of map file

```
custom "exp(-(v_E_a-rxnbond(c_bondforce,bond1frag)*v_l0)/(2/3*rxnave(v_ke))) < random(0,1,12345)
→"
```

By using an inequality and the ‘random(x,y,z)’ function, the left-hand side can be interpreted as the probability of the reaction occurring, similar to the ‘arrhenius’ constraint above.

By default, all constraints must be satisfied for the reaction to occur. In other words, constraints are evaluated as a series of logical values using the logical AND operator “`&&`”. More complex logic can be achieved by explicitly adding the logical AND operator “`&&`” or the logical OR operator “`||`” after a given constraint command. If a logical operator is specified after a constraint, it must be placed after all constraint parameters, on the same line as the constraint (one per line). Similarly, parentheses can be used to group constraints. The expression that results from concatenating all constraints should be a valid logical expression that can be read by the `variable` command after converting each constraint to a logical value. Because exactly one constraint is allowed per line, having a valid logical expression implies that left parentheses “`(`” should only appear before a constraint, and right parentheses “`)`” should only appear after a constraint and before any logical operator.

Once a reaction site has been successfully identified, data structures within LAMMPS that store bond topology are updated to reflect the post-reacted molecule template. All force fields with fixed bonds, angles, dihedrals or impropers are supported.

A few capabilities to note:

- (1) You may specify as many `react` arguments as desired. For example, you could break down a complicated reaction mechanism into several reaction steps, each defined by its own `react` argument.
- (2) While typically a bond is formed or removed between the initiator atoms specified in the pre-reacted molecule template, this is not required.
- (3) By reversing the order of the pre- and post-reacted molecule templates in another `react` argument, you can allow for the possibility of one or more reverse reactions.

The optional keywords deal with the probability of a given reaction occurring as well as the stable equilibration of each reaction site as it occurs.

The `prob` keyword can affect whether or not an eligible reaction actually occurs. The fraction setting must be a value between 0.0 and 1.0, and can be specified with an equal-style `variable`. A uniform random number between 0.0 and 1.0 is generated and the eligible reaction only occurs if the random number is less than the fraction. Up to `N` reactions are permitted to occur, as optionally specified by the `max_rxn` keyword.

Added in version 22Dec2022.

The `rate_limit` keyword can enforce an upper limit on the overall rate of the reaction. The number of reaction occurrences is limited to `Nlimit` within an interval of `Nsteps` timesteps. No reactions are permitted to occur within the first `Nsteps` timesteps of the first run after reading a data file. `Nlimit` can be specified with an equal-style `variable`.

The `stabilize_steps` keyword allows for the specification of how many time steps a reaction site is stabilized before being returned to the overall system thermostat. In order to produce the most physical behavior, this “reaction site equilibration time” should be tuned to be as small as possible while retaining stability for a given system or reaction step. After a limited number of case studies, this number has been set to a default of 60 time steps. Ideally, it should be individually tuned for each fix reaction step. Note that in some situations, decreasing rather than increasing this parameter will result in an increase in stability.

The *custom_charges* keyword can be used to specify which atoms' atomic charges are updated. When the value is set to *no*, all atomic charges are updated to those specified by the post-reaction template (default). Otherwise, the value should be the name of a molecule fragment defined in the pre-reaction molecule template. In this case, only the atomic charges of atoms in the molecule fragment are updated.

Added in version 22Dec2022.

The *rescale_charges* keyword can be used to ensure the total charge of the system does not change as reactions occur. When the argument is set to *yes*, a fixed value is added to the charges of post-reaction atoms such that their total charge equals that of the pre-reaction site. If only a subset of atomic charges are updated via the *custom_charges* keyword, this rescaling is applied to the subset. This keyword could be useful for systems that contain different molecules with the same reactive site, if the partial charges on the reaction site vary from molecule to molecule, or when removing reaction by-products.

The *molecule* keyword can be used to force the reaction to be intermolecular, intramolecular or either. When the value is set to *off*, molecule IDs are not considered when searching for reactions (default). When the value is set to *inter*, the initiator atoms must have different molecule IDs in order to be considered for the reaction. When the value is set to *intra*, only initiator atoms with the same molecule ID are considered for the reaction.

A few other considerations:

Optionally, you can enforce additional behaviors on reacting atoms. For example, it may be beneficial to force reacting atoms to remain at a certain temperature. For this, you can use the internally-created dynamic group named “*bond_react_MASTER_group*”, which consists of all atoms currently involved in a reaction. For example, adding the following command would add an additional thermostat to the group of all currently-reacting atoms:

```
fix 1 bond_react_MASTER_group temp/rescale 1 300 300 10 1
```

Note

This command must be added after the fix bond/react command, and will apply to all reactions.

Computationally, each time step this fix is invoked, it loops over neighbor lists (for bond-forming reactions) and computes distances between pairs of atoms in the list. It also communicates between neighboring processors to coordinate which bonds are created and/or removed. All of these operations increase the cost of a time step. 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.

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

Cumulative reaction counts for each reaction are written to *binary restart files*. These values are associated with the reaction name (react-ID). Additionally, internally-created per-atom properties are stored to allow for smooth restarts. None of the *fix_modify* options are relevant to this fix.

This fix computes one statistic for each *react* argument that it stores in a global vector, of length (number of react arguments), that can be accessed by various *output commands*. The vector values calculated by this fix are “intensive”.

There is one quantity in the global vector for each *react* argument:

- (1) cumulative number of reactions that occurred

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*.

When fix bond/react is “*unfixed*”, all internally-created groups are deleted. Therefore, fix bond/react can only be unfixed after unfixing all other fixes that use any group created by fix bond/react.

2.30.5 Restrictions

This fix is part of the REACTION package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

2.30.6 Related commands

fix bond/create, fix bond/break, fix bond/swap, dump local, special_bonds

2.30.7 Default

The option defaults are stabilization = no, prob = 1.0, stabilize_steps = 60, reset_mol_ids = yes, custom_charges = no, molecule = off, modify_create = *fit all*

(Gissinger2017) Gissinger, Jensen and Wise, Polymer, 128, 211-217 (2017).

(Gissinger2020) Gissinger, Jensen and Wise, Macromolecules, 53, 22, 9953-9961 (2020).

(Gissinger2024) Gissinger, Jensen and Wise, Computer Physics Communications, 304, 109287 (2024).

2.31 fix bond/swap command

2.31.1 Syntax

```
fix ID group-ID bond/swap Nevery fraction cutoff seed
```

- ID, group-ID are documented in [fix](#) command
- bond/swap = style name of this fix command
- Nevery = attempt bond swapping every this many steps
- fraction = fraction of group atoms to consider for swapping
- cutoff = distance at which swapping will be considered (distance units)
- seed = random # seed (positive integer)

2.31.2 Examples

```
fix 1 all bond/swap 50 0.5 1.3 598934
```

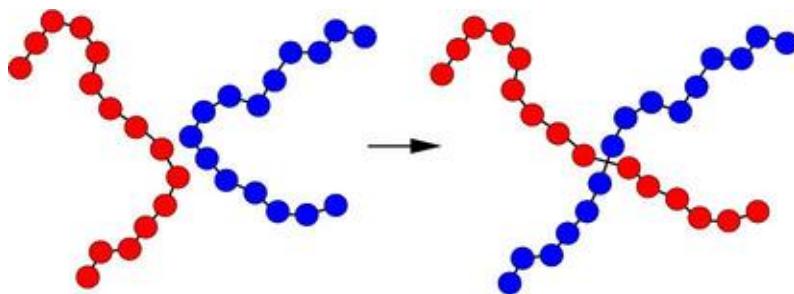
2.31.3 Description

In a simulation of polymer chains this command attempts to swap a pair of bonds, as illustrated below. This is done via Monte Carlo rules using the Boltzmann acceptance criterion, typically with the goal of equilibrating the polymer system more quickly. This fix is designed for use with idealized bead-spring polymer chains where each polymer is a linear chain of monomers, but LAMMPS does not check that is the case for your system.

Here are two use cases for this fix.

The first use case is for swapping bonds on two different chains, effectively grafting the end of one chain onto the other chain and vice versa. The purpose is to equilibrate the polymer chain conformations more rapidly than dynamics alone would do it, by enabling instantaneous large conformational changes in a dense polymer melt. The polymer chains should thus more rapidly converge to the proper end-to-end distances and radii of gyration.

A schematic of the kinds of bond swaps that can occur in this use case is shown here:



On the left, the red and blue chains have two monomers A1 and B1 close to each other, which are currently bonded to monomers A2 and B2 respectively within their own chains. The bond swap operation will attempt to delete the A1-A2 and B1-B2 bonds and replace them with A1-B2 and B1-A2 bonds. If the swap is energetically favorable, the two chains on the right are the result and each polymer chain has undergone a dramatic conformational change. This reference, ([Sides](#)) provides more details on the algorithm's effectiveness for this use case.

The second use case is a collection of polymer chains with some fraction of their sites identified as “sticker” sites. Initially each polymer chain is isolated from the others in a topological sense, and there is an intra-chain bond between every pair of sticker sites on the same chain. Over time, bonds swap so that inter-molecular sticker bonds are created. This models a vitrification-style process whereby the polymer chains all become interconnected. For this use case, if angles are defined they should not include bonds between sticker sites.

Note

For the sticker site model, you should set the newton flag for bonds to “off”, via the `newton on off` command (“on” is the default for the 2nd argument). This is to ensure appropriate randomness in bond selection because the I,J bond will be stored by both atom I and atom J. LAMMPS cannot check for this, because it is not aware that a sticker site model is being used.

The bond swapping operation is invoked once every *N*every timesteps. If any bond in the entire system is swapped, a re-build of the neighbor lists is triggered, since a swap alters the list of which neighbors are considered for pairwise interaction. At each invocation, each processor considers a random specified *fraction* of its atoms as potential swapping monomers for this timestep. Choosing a small *fraction* value can reduce the likelihood of a reverse swap occurring soon after an initial swap.

For each monomer A1, its neighbors are looped over as B1 monomers. For each A1,B1 an additional double loop of bond partners A2 of A1, and bond partners B2 of B1 a is performed. For each pair of A1-A2 and B1-B2 bonds to be eligible for swapping, the following 4 criteria must be met:

1. All 4 monomers must be in the fix group.
2. All 4 monomers must be owned by the processor (not ghost atoms). This ensures that another processor does not attempt to swap bonds involving the same atoms on the same timestep. Note that this also means that bond pairs which straddle processor boundaries are not eligible for swapping on this step.
3. The distances between 4 pairs of atoms –(A1,A2), (B1,B2), (A1,B2), (B1,A2)– must all be less than the specified *cutoff*.
4. The molecule IDs of A1 and B1 must be the same (see below).

If an eligible B1 partner is found, the energy change due to swapping the two bonds is computed. This includes changes in pairwise, bond, and angle energies due to the altered connectivity of the 2 chains. Dihedral and improper interactions are not allowed to be defined when this fix is used.

If the energy decreases due to the swap operation, the bond swap is accepted. If the energy increases it is accepted with probability $\exp(-\Delta E/kT)$ where ΔE is the increase in energy, k is the Boltzmann constant, and T is the current temperature of the system.

Note

Whether the swap is accepted or rejected, no other swaps are attempted by this processor on this timestep. No other eligible 4-tuples of atoms are considered. This means that each processor will perform either a single swap or none on timesteps this fix is invoked.

The criterion for matching molecule IDs is how the first use case described above can be simulated while conserving chain lengths. This is done by setting up the molecule IDs for the polymer chains in a specific way, typically in the data file, read by the [read_data](#) command.

Consider a system of 6-mer chains. You have 2 choices. If the molecule IDs for monomers on each chain are set to 1,2,3,4,5,6 then swaps will conserve chain length. For a particular monomer there will be only one other monomer on another chain which is a potential swap partner. If the molecule IDs for monomers on each chain are set to 1,2,3,3,2,1 then swaps will conserve chain length but swaps will be able to occur at either end of a chain. Thus for a particular monomer there will be 2 possible swap partners on another chain. In this scenario, swaps can also occur within a single chain, i.e. the two ends of a chain swap with each other.

Note

If your simulation uses molecule IDs in the usual way, where all monomers on a single chain are assigned the same ID (different for each chain), then swaps will only occur within the same chain. If you assign the same molecule ID to all monomers in all chains then inter-chain swaps will occur, but they will not conserve chain length. Neither of these scenarios is probably what you want for this fix.

Note

When a bond swap occurs the image flags of monomers in the new polymer chains can become inconsistent. See the [dump](#) command for a discussion of image flags. This is not an issue for running dynamics, but can affect calculation of some diagnostic quantities or the printing of unwrapped coordinates to a dump file.

For the second use case described above, the molecule IDs for all sticker sites should be the same.

This fix computes a temperature each time it is invoked for use by the Boltzmann criterion. To do this, the fix creates its own compute of style *temp*, as if this command had been issued:

```
compute fix-ID_temp all temp
```

See the [compute temp](#) command for details. Note that the ID of the new compute is the fix-ID with underscore + “temp” appended and the group for the new compute is “all”, so that the temperature of the entire system is used.

Note that this is NOT the compute used by thermodynamic output (see the [thermo_style](#) command) with ID = *thermo_temp*. This means you can change the attributes of this fix’s temperature (e.g. its degrees-of-freedom) via the [compute_modify](#) command or print this temperature during thermodynamic output via the [thermo_style custom](#) command using the appropriate compute-ID. It also means that changing attributes of *thermo_temp* will have no effect on this fix.

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

No information about this fix is written to [binary restart files](#). Because the state of the random number generator is not saved in restart files, this means you cannot do “exact” restarts with this fix, where the simulation continues on the same as if no restart had taken place. However, in a statistical sense, a restarted simulation should produce the same behavior. Also note that each processor generates possible swaps independently of other processors. Thus if you repeat the same simulation on a different number of processors, the specific swaps performed will be different.

The [fix_modify temp](#) option is supported by this fix. You can use it to assign a [compute](#) you have defined to this fix which will be used to compute the temperature for the Boltzmann criterion.

This fix computes two statistical quantities as a global 2-vector of output, which can be accessed by various [output commands](#). The first component of the vector is the cumulative number of swaps performed by all processors. The second component of the vector is the cumulative number of swaps attempted (whether accepted or rejected). Note that a swap “attempt” only occurs when swap partners meeting the criteria described above are found on a particular timestep. The vector values calculated by this fix are “intensive”.

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](#).

2.31.5 Restrictions

This fix is part of the MC package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) doc page for more info.

The settings of the “special_bond” command must be 0,1,1 in order to use this fix, which is typical of bead-spring chains with FENE or harmonic bonds. This means that pairwise interactions between bonded atoms are turned off, but are turned on between atoms two or three hops away along the chain backbone.

Currently, energy changes in dihedral and improper interactions due to a bond swap are not considered. Thus a simulation that uses this fix cannot use a dihedral or improper potential.

2.31.6 Related commands

fix atom/swap

2.31.7 Default

none

(Sides) Sides, Grest, Stevens, Plimpton, J Polymer Science B, 42, 199-208 (2004).

2.32 fix box/relax command

2.32.1 Syntax

```
fix ID group-ID box/relax keyword value ...
```

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

one or more keyword value pairs may be appended

keyword = iso or aniso or tri or x or y or z or xy or yz or xz or couple or nreset or vmax or dilate

→ or scaleyz or scalexz or scalexy or fixedpoint

iso or aniso or tri value = Ptargt = desired pressure (pressure units)

x or y or z or xy or yz or xz value = Ptargt = 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)

2.32.2 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
```

2.32.3 Description

Apply an external pressure or stress tensor to the simulation box during an [energy minimization](#). This allows the box size and shape to vary during the iterations of the minimizer so that the final configuration will be both an energy minimum for the potential energy of the atoms, and the system pressure tensor will be close to the specified external tensor. Conceptually, specifying a positive pressure is like squeezing on the simulation box; a negative pressure typically allows the box to expand.

The external pressure tensor is specified using one or more of the *iso*, *aniso*, *tri*, *x*, *y*, *z*, *xy*, *xz*, *yz*, and *couple* keywords. These keywords give you the ability to specify all 6 components of an external stress tensor, and to couple various of these components together so that the dimensions they represent are varied together during the minimization.

Orthogonal simulation boxes have 3 adjustable dimensions (x,y,z). Triclinic (non-orthogonal) simulation boxes have 6 adjustable dimensions (x,y,z,xy,xz,yz). The [create_box](#), [read data](#), and [read_restart](#) commands specify whether the simulation box is orthogonal or non-orthogonal (triclinic) and explain the meaning of the xy,xz,yz tilt factors.

The target pressures *Ptarget* for each of the 6 components of the stress tensor can be specified independently via the *x*, *y*, *z*, *xy*, *xz*, *yz* keywords, which correspond to the 6 simulation box dimensions. For example, if the *y* keyword is used, the *y*-box length will change during the minimization. If the *xy* keyword is used, the *xy* tilt factor will change. A box dimension will not change if that component is not specified.

Note that in order to use the *xy*, *xz*, or *yz* keywords, the simulation box must be triclinic, even if its initial tilt factors are 0.0.

When the size of the simulation box changes, all atoms are re-scaled to new positions, unless the keyword *dilate* is specified with a value of *partial*, in which case only the atoms in the fix group are re-scaled. This can be useful for leaving the coordinates of atoms in a solid substrate unchanged and controlling the pressure of a surrounding fluid.

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 LAMMPS, 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.

Note

Applying 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, LAMMPS will run less efficiently, due to the large volume of communication needed to acquire ghost atoms around a processor's irregular-shaped subdomain. For extreme values of tilt, LAMMPS may also lose atoms and generate an error.

 **Note**

Performing a minimization with this fix is not a mathematically well-defined minimization problem. This is because the objective function being minimized changes if the box size/shape changes. In practice this means the minimizer can get “stuck” before you have reached the desired tolerance. The solution to this is to restart the minimizer from the new adjusted box size/shape, since that creates a new objective function valid for the new box size/shape. Repeat as necessary until the box size/shape has reached its new equilibrium.

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 P_{xx} and P_{zz} 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 Ptargt y Ptargt z Ptargt couple xyz
```

The keyword *aniso* means *x*, *y*, and *z* dimensions are controlled independently using the P_{xx} , P_{yy} , and P_{zz} 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 Ptargt y Ptargt z Ptargt 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 Ptargt y Ptargt z Ptargt 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:

$$E = U + P_t (V - V_0) + E_{strain}$$

where *U* is the system potential energy, *P_t* is the desired hydrostatic pressure, *V* and *V₀* 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 \mathbf{P} will satisfy the relation:

$$\mathbf{P} = P_t \mathbf{I} + \mathbf{S}_t (\mathbf{h}_0^{-1})^t \mathbf{h}_{0d}$$

where \mathbf{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 LAMMPS so that the upper-triangular components of \mathbf{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* 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.

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 pressure compute this command creates (see below) or a pressure compute you define yourself.

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, (b) minimize several times in succession if need be, to drive the pressure closer to the target pressure, (c) relax the atom positions before relaxing the box, and (d) relax the box to the target hydrostatic pressure before relaxing to a target shear stress state. 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.

2.32.4 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.

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 box/relax), 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, as described above. The energy values reported at the end of a minimization run under “Minimization stats” include this energy, and so differ from what LAMMPS normally reports as potential energy. This fix does not support the `fix_modify energy` option, because that would result in double-counting of the fix energy in the minimization energy. Instead, the fix energy can be explicitly added to the potential energy using one of these two variants:

```
variable emin equal pe+f_1
variable emin equal pe+f_1/atoms
```

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.

2.32.5 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 second dimension in the keyword is periodic, and if the tilt factor is not coupled to the barostat via keywords `tri`, `yz`, `xz`, and `xy`.

2.32.6 Related commands

fix npt, minimize

2.32.7 Default

The keyword defaults are dilate = all, vmax = 0.0001, nreset = 0.

(Parrinello1981) Parrinello and Rahman, J Appl Phys, 52, 7182 (1981).

2.33 fix brownian command

2.34 fix brownian/sphere command

2.35 fix brownian/asphere command

2.35.1 Syntax

```
fix ID group-ID style_name temp seed keyword args
```

- ID, group-ID are documented in [fix](#) command
- style_name = *brownian* or *brownian/sphere* or *brownian/asphere*
- temp = temperature
- seed = random number generator seed
- one or more keyword/value pairs may be appended
- keyword = *rng* or *dipole* or *gamma_r_eigen* or *gamma_t_eigen* or *gamma_r* or *gamma_t* or *rotation_temp* or *planar_rotation*

rng value = uniform or gaussian or none

uniform = use uniform random number generator

gaussian = use gaussian random number generator

none = turn off noise

dipole value = mux and muy and muz for brownian/asphere

mux, muy, and muz = update orientation of dipole having direction (mux,*muy*,*muz*) in body frame of rigid body

gamma_r_eigen values = gr1 and gr2 and gr3 for brownian/asphere

gr1, gr2, and gr3 = diagonal entries of body frame rotational friction tensor

gamma_r values = gr for brownian/sphere

gr = magnitude of the (isotropic) rotational friction tensor

gamma_t_eigen values = gt1 and gt2 and gt3 for brownian/asphere

gt1, gt2, and gt3 = diagonal entries of body frame translational friction tensor

gamma_t values = gt for brownian and brownian/sphere

gt = magnitude of the (isotropic) translational friction tensor

rotation_temp values = T for brownian/sphere and brownian/asphere

T = rotation temperature, which can be different than temp when out of equilibrium

planar_rotation values = none (constrains rotational diffusion to be in xy plane if in 3D)

2.35.2 Examples

```
fix 1 all brownian 1.0 12908410 gamma_t 1.0
fix 1 all brownian 1.0 12908410 gamma_t 3.0 rng gaussian
fix 1 all brownian/sphere 1.0 1294019 gamma_t 3.0 gamma_r 1.0
fix 1 all brownian/sphere 1.0 19581092 gamma_t 1.0 gamma_r 0.3 rng none
fix 1 all brownian/asphere 1.0 1294019 gamma_t_eigen 1.0 2.0 3.0 gamma_r_eigen 4.0 7.0 8.0 rng_
→gaussian
fix 1 all brownian/asphere 1.0 1294019 gamma_t_eigen 1.0 2.0 3.0 gamma_r_eigen 4.0 7.0 8.0 dipole 1.0_
→0.0 0.0
```

2.35.3 Description

Perform Brownian Dynamics time integration to update position, velocity, dipole orientation (for spheres) and quaternion orientation (for ellipsoids, with optional dipole update as well) of all particles in the fix group in each timestep. Brownian Dynamics uses Newton's laws of motion in the limit that inertial forces are negligible compared to viscous forces. The stochastic equation of motion for the center of mass positions is

$$d\mathbf{r} = \gamma_t^{-1} \mathbf{F} dt + \sqrt{2k_B T} \gamma_t^{-1/2} d\mathbf{W}_t,$$

in the lab-frame (i.e., γ_t is not diagonal, but only depends on orientation and so the noise is still additive).

The rotational motion for the spherical and ellipsoidal particles is not as simple an expression, but is chosen to replicate the Boltzmann distribution for the case of conservative torques (see ([Ilie](#)) or ([Delong](#))).

For the style *brownian*, only the positions of the particles are updated. This is therefore suitable for point particle simulations.

For the style *brownian/sphere*, the positions of the particles are updated, and a dipole slaved to the spherical orientation is also updated. This style therefore requires the hybrid atom style *atom_style dipole* and *atom_style sphere*. The equation of motion for the dipole is

$$\mu(t+dt) = \frac{\mu(t) + \omega \times \mu dt}{|\mu(t) + \omega \times \mu|}$$

which correctly reproduces a Boltzmann distribution of orientations and rotational diffusion moments (see ([Ilie](#))) when

$$\omega = \frac{\mathbf{T}}{\gamma_r} + \sqrt{\frac{2k_B T_{rot}}{\gamma_r} \frac{d\mathbf{W}}{dt}},$$

with $d\mathbf{W}$ being a random number with zero mean and variance dt and T_{rot} is *rotation_temp*.

For the style *brownian/asphere*, the center of mass positions and the quaternions of ellipsoidal particles are updated. This fix style is suitable for equations of motion where the rotational and translational friction tensors can be diagonalized in a certain (body) reference frame. In this case, the rotational equation of motion is updated via the quaternion

$$\mathbf{q}(t+dt) = \frac{\mathbf{q}(t) + d\mathbf{q}}{\|\mathbf{q}(t) + d\mathbf{q}\|}$$

which correctly reproduces a Boltzmann distribution of orientations and rotational diffusion moments [see ([Ilie](#))] when the quaternion step is given by

$$d\mathbf{q} = \Psi \omega dt$$

where Ψ has rows $(-q_1, -q_2, -q_3)$, $(q_0, -q_3, q_2)$, $(q_3, q_0, -q_1)$, and $(-q_2, q_1, q_0)$. ω is evaluated in the body frame of reference where the friction tensor is diagonal. See ([Delong](#)) for more details of a similar algorithm.

Note

This integrator does not by default assume a relationship between the rotational and translational friction tensors, though such a relationship should exist in the case of no-slip boundary conditions between the particles and the surrounding (implicit) solvent. For example, in the case of spherical particles, the condition $\gamma_t = 3\gamma_r/\sigma^2$ must be explicitly accounted for by setting *gamma_t* to 3x and *gamma_r* to x (where σ is the sphere's diameter). A similar (though more complex) relationship holds for ellipsoids and rod-like particles. The translational diffusion and rotational diffusion are given by *temp/gamma_t* and *rotation_temp/gamma_r*.

Note

Temperature computation using the *compute temp* will not correctly compute the temperature of these overdamped dynamics since we are explicitly neglecting inertial effects. Furthermore, this time integrator does not add the stochastic terms or viscous terms to the force and/or torques. Rather, they are just added in to the equations of motion to update the degrees of freedom.

If the *rng* keyword is used with the *uniform* value, then the noise is generated from a uniform distribution (see ([Dunweg](#)) for why this works). This is the same method of noise generation as used in *fix_langevin*.

If the *rng* keyword is used with the *gaussian* value, then the noise is generated from a Gaussian distribution. Typically this added complexity is unnecessary, and one should be fine using the *uniform* value for reasons argued in ([Dunweg](#)).

If the *rng* keyword is used with the *none* value, then the noise terms are set to zero.

The *gamma_t* keyword sets the (isotropic) translational viscous damping. Required for (and only compatible with) *brownian* and *brownian/sphere*. The units of *gamma_t* are mass/time.

The *gamma_r* keyword sets the (isotropic) rotational viscous damping. Required for (and only compatible with) *brownian* and *brownian/sphere*. The units of *gamma_r* are mass*length**2/time.

The *gamma_r_eigen*, and *gamma_t_eigen* keywords are the eigenvalues of the rotational and viscous damping tensors (having the same units as their isotropic counterparts). Required for (and only compatible with) *brownian/asphere*. For a 2D system, the first two values of *gamma_r_eigen* must be *inf* (only rotation in x–y plane), and the third value of *gamma_t_eigen* must be *inf* (only diffusion in the x–y plane).

If the *dipole* keyword is used, then the dipole moments of the particles are updated as described above. Only compatible with *brownian/asphere* (as *brownian/sphere* updates dipoles automatically).

If the *rotation_temp* keyword is used, then the rotational diffusion will be occur at this prescribed temperature instead of *temp*. Only compatible with *brownian/sphere* and *brownian/asphere*.

If the *planar_rotation* keyword is used, then rotation is constrained to the x–y plane in a 3D simulation. Only compatible with *brownian/sphere* and *brownian/asphere* in 3D.

Note

For style *brownian/asphere*, the components *gamma_t_eigen* = (x,x,x) and *gamma_r_eigen* = (y,y,y), the dynamics will replicate those of the *brownian/sphere* style with *gamma_t* = x and *gamma_r* = y.

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

No information about this fix is written to *binary restart files*. 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*.

2.35.5 Restrictions

The style *brownian/sphere* fix requires that atoms store torque and angular velocity (*omega*) as defined by the *atom_style sphere* command. The style *brownian/asphere* fix requires that atoms store torque and quaternions as defined by the *atom_style ellipsoid* command. If the *dipole* keyword is used, they must also store a dipole moment as defined by the *atom_style dipole* command.

This fix is part of the BROWNIAN package. It is only enabled if LAMMPS was built with that package. See the *Build package* doc page for more info.

2.35.6 Related commands

fix propel/self, *fix langevin*, *fix nve/sphere*,

2.35.7 Default

The default for *rng* is *uniform*. The default for the rotational and translational friction tensors are the identity tensor.

(Ilie) Ilie, Briels, den Otter, Journal of Chemical Physics, 142, 114103 (2015).

(Delong) Delong, Usabiaga, Donev, Journal of Chemical Physics. 143, 144107 (2015)

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

2.36 fix charge/regulation command

2.36.1 Syntax

```
fix ID group-ID charge/regulation cation_type anion_type keyword value(s)
```

- ID, group-ID are documented in fix command
 - charge/regulation = style name of this fix command
 - cation_type = atom type of free cations (integer or type label)
 - anion_type = atom type of free anions (integer or type label)
 - zero or more keyword/value pairs may be appended
- keyword = pH, pKa, pKb, pIp, pIm, pKs, acid_type, base_type, lunit_nm, temp, tempfixid,
→nevery, nmc, rxd, seed, tag, group, onlysalt, pmcmoves
pH value = pH of the solution (can be specified as an equal-style variable)
pKa value = acid dissociation constant (in the -log10 representation)

pKb value = base dissociation constant (in the -log10 representation)
 pIp value = activity (effective concentration) of free cations (in the -log10 representation)
 pIm value = activity (effective concentration) of free anions (in the -log10 representation)
 pKs value = solvent self-dissociation constant (in the -log10 representation)
 acid_type = atom type of acid groups (integer or type label)
 base_type = atom type of base groups (integer or type label)
 lunit_nm value = unit length used by LAMMPS (# in the units of nanometers)
 temp value = temperature
 tempfixid value = fix ID of temperature thermostat
 nevery value = invoke this fix every nevery steps
 nmc value = number of charge regulation MC moves to attempt every nevery steps
 rxd value = cutoff distance for acid/base reaction
 seed value = random # seed (positive integer)
 tag value = yes or no (yes: The code assign unique tags to inserted ions; no: The tag of all inserted ions is "0")
 group value = group-ID, inserted ions are assigned to group group-ID. Can be used multiple times to assign inserted ions to multiple groups.
 onlysalt values = flag charge_cation charge_anion.
 flag = yes or no (yes: the fix performs only ion insertion/deletion, no: perform acid/base dissociation and ion insertion/deletion)
 charge_cation, charge_anion = value of cation/anion charge, must be an integer (only specify if flag = yes)
 pmcmoves values = pmcA pmcB pmcI - MC move fractions for acid ionization (pmcA), base ionization (pmcB) and free ion exchange (pmcI)

2.36.2 Examples

```

fix chareg all charge/regulation 1 2 acid_type 3 base_type 4 pKa 5.0 pKb 6.0 pH 7.0 pIp 3.0 pIm 3.0
  ↵nevery 200 nmc 200 seed 123 tempfixid fT
fix chareg all charge/regulation 1 2 pIp 3 pIm 3 onlysalt yes 2 -1 seed 123 tag yes temp 1.0

labelmap atom 1 H+ 2 OH-
fix chareg all charge/regulation H+ OH- pIp 3 pIm 3 onlysalt yes 2 -1 seed 123 tag yes temp 1.0

```

2.36.3 Description

This fix performs Monte Carlo (MC) sampling of charge regulation and exchange of ions with a reservoir as discussed in ([Cukr1](#)) and ([Cukr2](#)). The implemented method is largely analogous to the grand-reaction ensemble method in ([Landsgesell](#)). The implementation is parallelized, compatible with existing LAMMPS functionalities, and applicable to any system utilizing discrete, ionizable groups or surface sites.

Specifically, the fix implements the following three types of MC moves, which discretely change the charge state of individual particles and insert ions into the systems: $A \rightleftharpoons A^- + X^+$, $B \rightleftharpoons B^+ + X^-$, and $\emptyset \rightleftharpoons Z^-X^{Z^+} + Z^+X^{-Z^-}$. In the former two types of reactions, Monte Carlo moves alter the charge value of specific atoms (A, B) and simultaneously insert a counterion to preserve the charge neutrality of the system, modeling the dissociation/association process. The last type of reaction performs grand canonical MC exchange of ion pairs with a (fictitious) reservoir.

In our implementation “acid” refers to particles that can attain charge $q = \{0, -1\}$ and “base” to particles with $q = \{0, 1\}$, whereas the MC exchange of free ions allows any integer charge values of Z^+ and Z^- .

Here we provide several practical examples for modeling charge regulation effects in solvated systems. An acid ionization reaction ($A \rightleftharpoons A^- + H^+$) can be defined via a single line in the input file

```
fix acid_reaction all charge/regulation 2 3 acid_type 1 pH 7.0 pKa 5.0 pIp 7.0 pIm 7.0
```

where the fix attempts to charge A (discharge A^-) to A^- (A) and insert (delete) a proton (atom type 2). Besides, the fix implements self-ionization reaction of water $\emptyset \rightleftharpoons H^+ + OH^-$.

However, this approach is highly inefficient at $pH \approx 7$ when the concentration of both protons and hydroxyl ions is low, resulting in a relatively low acceptance rate of MC moves.

A more efficient way is to allow salt ions to participate in ionization reactions, which can be easily achieved via

```
fix acid_reaction2 all charge/regulation 4 5 acid_type 1 pH 7.0 pKa 5.0 pIp 2.0 pIm 2.0
```

where particles of atom type 4 and 5 are the salt cations and anions, both at activity (effective concentration) of 10^{-2} mol/l, see ([Cerk1](#)) and ([Landsgesell](#)) for more details.

We could have simultaneously added a base ionization reaction ($B \rightleftharpoons B^+ + OH^-$)

```
fix acid_base_reaction all charge/regulation 2 3 acid_type 1 base_type 6 pH 7.0 pKa 5.0 pKb 6.0 pIp 7.  
→ 0 pIm 7.0
```

where the fix will attempt to charge B (discharge B^+) to B^+ (B) and insert (delete) a hydroxyl ion OH^- of atom type 3.

Dissociated ions and salt ions can be combined into a single particle type, which reduces the number of necessary MC moves and increases sampling performance, see ([Cerk1](#)). The H^+ and monovalent salt cation (S^+) are combined into a single particle type, $X^+ = \{H^+, S^+\}$. In this case “pIp” refers to the effective concentration of the combined cation type X^+ and its value is determined by $10^{-pIp} = 10^{-pH} + 10^{-pSp}$, where 10^{-pSp} is the effective concentration of salt cations. For example, at $pH=7$ and $pSp=6$ we would find $pIp \sim 5.958$ and the command that performs reactions with combined ions could read,

```
fix acid_reaction_combined all charge/regulation 2 3 acid_type 1 pH 7.0 pKa 5.0 pIp 5.958 pIm 5.958
```

If neither the acid or the base type is specified, for example,

```
fix salt_reaction all charge/regulation 4 5 pIp 2.0 pIm 2.0
```

the fix simply inserts or deletes an ion pair of a free cation (atom type 4) and a free anion (atom type 5) as done in a conventional grand-canonical MC simulation. Multivalent ions can be inserted (deleted) by using the *onlysalt* keyword.

This fix is compatible with LAMMPS packages such as MOLECULE or RIGID. The acid and base particles can be part of larger molecules or rigid bodies. Free ions that are inserted to or deleted from the system must be defined as single particles (no bonded interactions allowed) and cannot be part of larger molecules or rigid bodies. If an atom style with molecule IDs is used, all inserted ions have a molecule ID equal to zero.

Note that LAMMPS implicitly assumes a constant number of particles (degrees of freedom). Since using this fix alters the total number of particles during the simulation, any thermostat used by LAMMPS, such as NVT or Langevin, must use a dynamic calculation of system temperature. This can be achieved by specifying a dynamic temperature compute (e.g. dtemp) and using it with the desired thermostat, e.g. a Langevin thermostat:

```
compute dtemp all temp
compute_modify dtemp dynamic/dof yes
fix fT all langevin 1.0 1.0 1.0 123
fix_modify fT temp dtemp
```

The units of pH, pKa, pKb, pIp, pIm are considered to be in the standard -log10 representation assuming reference concentration $\rho_0 = \text{mol/l}$. For example, in the dilute ideal solution limit, the concentration of free cations will be $c_1 = 10^{-pIp} \text{ mol/l}$. To perform the internal unit conversion, the the value of the LAMMPS unit length must be specified in nanometers via *lunit_nm*. The default value is set to the Bjerrum length in water at room temperature (0.71 nm), *lunit_nm* = 0.71.

The temperature used in MC acceptance probability is set by *temp*. This temperature should be the same as the temperature set by the molecular dynamics thermostat. For most purposes, it is probably best to use *tempfixid* keyword which dynamically sets the temperature equal to the chosen MD thermostat temperature, in the example above we assumed the thermostat fix-ID is *ft*. The inserted particles attain a random velocity corresponding to the specified temperature. Using *tempfixid* overrides any fixed temperature set by *temp*.

The *rxr* keyword can be used to restrict the inserted/deleted counterions to a specific radial distance from an acid or base particle that is currently participating in a reaction. This can be used to simulate more realistic reaction dynamics. If *rxr* = 0 or *rxr* > *L* / 2, where *L* is the smallest box dimension, the radial restriction is automatically turned off and free ion can be inserted or deleted anywhere in the simulation box.

If the *tag yes* is used, every inserted atom gets a unique tag ID, otherwise, the tag of every inserted atom is set to 0. *tag yes* might cause an integer overflow in very long simulations since the tags are unique to every particle and thus increase with every successful particle insertion.

The *pmcmoves* keyword sets the relative probability of attempting the three types of MC moves (reactions): acid charging, base charging, and ion pair exchange. The fix only attempts to perform particle charging MC moves if *acid_type* or *base_type* is defined. Otherwise fix only performs free ion insertion/deletion. For example, if *acid_type* is not defined, *pmcA* is automatically set to 0. The vector *pmcmoves* is automatically normalized, for example, if set to *pmcmoves* 0 0.33 0.33, the vector would be normalized to [0,0.5,0.5].

The *only_salt* option can be used to perform multivalent grand-canonical ion-exchange moves. If *only_salt yes* is used, no charge exchange is performed, only ion insertion/deletion (*pmcmoves* is set to [0,0,1]), but ions can be multivalent. In the example above, an MC move would consist of three ion insertion/deletion to preserve the charge neutrality of the system.

The *group* keyword can be used to add inserted particles to a specific group-ID. All inserted particles are automatically added to group *all*.

2.36.4 Output

This fix computes a global vector of length 8, which can be accessed by various output commands. The vector values are the following global quantities:

1. cumulative MC attempts
2. cumulative MC successes
3. current # of neutral acid atoms
4. current # of -1 charged acid atoms
5. current # of neutral base atoms
6. current # of +1 charged base atoms
7. current # of free cations
8. current # of free anions

2.36.5 Restrictions

This fix is part of the MC package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

The [atom_style](#), used must contain the charge property and have per atom type masses, for example, the style could be *charge* or *full*. Only usable for 3D simulations. Atoms specified as free ions cannot be part of rigid bodies or molecules and cannot have bonding interactions. The scheme is limited to integer charges, any atoms with non-integer charges will not be considered by the fix.

All interaction potentials used must be continuous, otherwise the MD integration and the particle exchange MC moves do not correspond to the same equilibrium ensemble. For example, if an *lj/cut* pair style is used, the LJ potential must be shifted so that it vanishes at the cutoff. This can be easily achieved using the [pair_modify](#) command, i.e., by using: *pair_modify shift yes*.

Note

Region restrictions are not yet implemented.

2.36.6 Related commands

[fix gcmc](#), [fix atom/swap](#)

2.36.7 Default

pH = 7.0; pKa = 100.0; pKb = 100.0; pIp = 5.0; pIm = 5.0; pKs = 14.0; acid_type = -1; base_type = -1; lunit_nm = 0.71; temp = 1.0; nevery = 100; nmc = 100; rxd = 0; seed = 0; tag = no; onlysalt = no, pmcmoves = [1/3, 1/3, 1/3], group-ID = all

(Curk1) T. Curk, J. Yuan, and E. Luijten, “Accelerated simulation method for charge regulation effects”, *JCP* 156 (2022).

(Curk2) T. Curk and E. Luijten, “Charge-regulation effects in nanoparticle self-assembly”, *PRL* 126 (2021)

(Landsgesell) J. Landsgesell, P. Hebbeker, O. Rud, R. Lunkad, P. Kosovan, and C. Holm, “Grand-reaction method for simulations of ionization equilibria coupled to ion partitioning”, *Macromolecules* 53, 3007-3020 (2020).

2.37 fix cmap command

2.37.1 Syntax

`fix ID group-ID cmap filename`

- ID, group-ID are documented in [fix](#) command
- cmap = style name of this fix command
- filename = force-field file with CMAP coefficients

2.37.2 Examples

```
fix      myCMAP all cmap ..../potentials/cmap36.data
read_data  proteinX.data fix myCMAP crossterm CMAP
fix_modify myCMAP energy yes
```

2.37.3 Description

This command enables CMAP 5-body interactions to be added to simulations which use the CHARMM force field. These are relevant for any CHARMM model of a peptide or protein sequences that is 3 or more amino-acid residues long; see ([Buck](#)) and ([Brooks](#)) for details, including the analytic energy expressions for CMAP interactions. The CMAP 5-body terms add additional potential energy contributions to pairs of overlapping phi-psi dihedrals of amino-acids, which are important to properly represent their conformational behavior.

The examples/cmap directory has a sample input script and data file for a small peptide, that illustrates use of the fix cmap command.

As in the example above, this fix should be used before reading a data file that contains a listing of CMAP interactions. The *filename* specified should contain the CMAP parameters for a particular version of the CHARMM force field. Two such files are including in the lammps/potentials directory: charmm22.cmap and charmm36.cmap.

The data file read by the “read_data” must contain the topology of all the CMAP interactions, similar to the topology data for bonds, angles, dihedrals, etc. Specially it should have a line like this in its header section:

N crossterms

where N is the number of CMAP 5-body interactions. It should also have a section in the body of the data file like this with N lines:

CMAP						
1	1	8	10	12	18	20
2	5	18	20	22	25	27
[...]						
N	3	314	315	317	318	330

The first column is an index from 1 to N to enumerate the CMAP 5-atom tuples; it is ignored by LAMMPS. The second column is the “type” of the interaction; it is an index into the CMAP force field file. The remaining 5 columns are the atom IDs of the atoms in the two 4-atom dihedrals that overlap to create the CMAP interaction. Note that the “crossterm” and “CMAP” keywords for the header and body sections match those specified in the read_data command following the data file name; see the [read_data](#) page for more details.

A data file containing CMAP 5-body interactions can be generated from a PDB file using the charmm21lammps.pl script in the tools/ch21lmp directory of the LAMMPS distribution. The script must be invoked with the optional “-cmap” flag to do this; see the tools/ch21lmp/README file for more information. The same conversion script also creates the file of CMAP coefficient data which is read by this command.

The potential energy associated with CMAP interactions can be output as described below. It can also be included in the total potential energy of the system, as output by the [thermo_style](#) command, if the [fix_modify energy](#) command is used, as in the example above. See the note below about how to include the CMAP energy when performing an [energy minimization](#).

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

This fix writes the list of CMAP cross-terms to *binary restart files*. See the [read_restart](#) command for info on how to re-specify a fix in an input script that reads a restart file, so that the operation of the fix continues in an uninterrupted fashion.

The [fix_modify energy](#) option is supported by this fix to add the potential energy of the CMAP interactions to both the global potential energy and peratom potential energies of the system as part of [thermodynamic output](#) or output by the [compute pe/atom](#) command. The default setting for this fix is [fix_modify energy yes](#).

The [fix_modify virial](#) option is supported by this fix to add the contribution due to the CMAP interactions to both the global pressure and per-atom stress of the system via the [compute pressure](#) and [compute stress/atom](#) commands. The former can be accessed by [thermodynamic output](#). The default setting for this fix is [fix_modify virial yes](#).

This fix computes a global scalar which can be accessed by various [output commands](#). The scalar is the potential energy discussed above. The scalar value calculated by this fix is “extensive”.

No parameter of this fix can be used with the *start/stop* keywords of the [run](#) command.

The forces due to this fix are imposed during an energy minimization, invoked by the [minimize](#) command.

The [fix_modify respa](#) option is supported by this fix. This allows to set at which level of the *r-RESPA* integrator the fix is adding its forces. Default is the outermost level.

Note

For energy minimization, if you want the potential energy associated with the CMAP terms forces to be included in the total potential energy of the system (the quantity being minimized), you MUST not disable the [fix_modify energy](#) option for this fix.

2.37.5 Restrictions

To function as expected this fix command must be issued *before* a [read_data](#) command but *after* a [read_restart](#) command.

This fix can only be used if LAMMPS was built with the MOLECULE package. See the [Build package](#) page for more info.

2.37.6 Related commands

[fix_modify](#), [read_data](#)

2.37.7 Default

none

(Buck) Buck, Bouquet-Bonnet, Pastor, MacKerell Jr., *Biophys J*, 90, L36 (2006).

(Brooks) Brooks, Brooks, MacKerell Jr., *J Comput Chem*, 30, 1545 (2009).

2.38 fix colvars command

2.38.1 Syntax

```
fix ID group-ID colvars *configfile* keyword value ...
```

- *ID, group-ID* are documented in [fix](#) command
- “colvars” = style name of this fix command
- *configfile* = configuration file for Colvars (use “*none*” to provide it inline)
- keyword = *output* or *input* or *unwrap* or *tstat* or *seed*
 - output* value = state filename/prefix for Colvars (default: “out”)
 - input* value = input state filename/prefix for Colvars (optional, default: “NULL”)
 - unwrap* value = “yes” or “no” (default: “yes”)
 - tstat* value = fix ID of thermostat applied to relevant atoms (default: “NULL”)
 - seed* value = seed for random number generator (default: 1966)

2.38.2 Examples

```
# Create the fix using a config file, set prefix for its output files
fix Colvars all colvars colvars.inp output ${JOB}

# Communicate the LAMMPS target temperature to the Colvars module
fix_modify Colvars tstat NPT

# Add a new restraint specific to this LAMMPS run
fix_modify Colvars config """
harmonic {
    name restraint
    colvars distance1 distance2
    centers ${ref1} ${ref2}
    forceConstant ${kappa}
}"""
```

2.38.3 Description

This fix interfaces LAMMPS to the collective variables [Colvars](#) library, which allows to accelerate sampling of rare events and the computation of free energy surfaces and potentials of mean force (PMFs) for any set of collective variables using a variety of sampling methods (e.g. umbrella-sampling, metadynamics, ABF...).

This documentation describes only the “fix colvars” command itself in a LAMMPS script. The Colvars library is fully documented in the included [PDF manual](#) or in the webpage <https://colvars.github.io/colvars-refman-lammps/colvars-refman-lammps.html>.

The Colvars library is developed at <https://github.com/Colvars/colvars>. A detailed discussion of its implementation is in ([Fiorin](#)); additional citations for specific features are printed at runtime if these features are used.

There are example scripts on the [Colvars website](#) as well as in the examples/PACKAGES/colvars directory in the LAMMPS source tree.

The only required argument to the fix is the name of the Colvars configuration file. The contents of this file are independent from the MD engine in which the Colvars library has been integrated, save for the units that are specific to each engine. In LAMMPS, the units used by Colvars are consistent with those specified by the [units](#) command.

Added in version Colvars_2023-06-04: The special value “*none*” (lowercase) initializes an empty Colvars module, which allows loading configuration dynamically using [fix_modify](#) (see below).

The *group-ID* entry is ignored. “fix colvars” will always apply to the entire system, but specific atoms will be selected based on selection keywords in the Colvars configuration file or files. There is no need to define multiple “fix colvars” instances and it is not allowed.

The “output” keyword allows to specify the prefix of output files generated by Colvars, for example “*output.colvars.traj*” or “*output.pmf*”. Supplying an empty string suppresses any file output from Colvars to file, except for data saved into the LAMMPS [binary restart](#) files.

The “input” keyword allows to specify an optional state file that contains the restart information needed to continue a previous simulation state. However, because “fix colvars” records its state in LAMMPS [binary restart](#) files, this is usually not needed when using the [read_restart](#) command.

The *unwrap* keyword controls whether wrapped or unwrapped coordinates are passed to the Colvars library for calculation of the collective variables and the resulting forces. The default is *yes*, i.e. the image flags are used to reconstruct the absolute atom positions. Setting this to *no* will use the current local coordinates that are wrapped back into the simulation cell at each re-neighboring step instead. For information about when and how this affects results, please see https://colvars.github.io/colvars-refman-lammps/colvars-refman-lammps.html#sec:colvar_atom_groups_wrapping.

The *tstat* keyword can be either “NULL” or the label of a thermostating fix that thermostats all atoms in the fix colvars group. This will be used to provide the colvars module with the current thermostat target temperature.

The *seed* keyword contains the seed for the random number generator that will be used in the colvars module.

2.38.4 Restarting

This fix writes the current state of the Colvars module into [binary restart files](#). This is in addition to the text-mode “.colvars.state” state file that is written by the Colvars module itself. The information contained in both files is identical, and the binary LAMMPS restart file is also used by fix colvars when [read_restart](#) is called in a LAMMPS script. In that case, there is typically no need to specify the *input* keyword.

As long as LAMMPS binary restarts will be used to continue a simulation, it is safe to delete the “.colvars.state” files to save space. However, when a LAMMPS simulation is restarted using [read_data](#), the Colvars state file must be available and loaded via the “input” keyword or via a “fix_modify Colvars load” command (see below).

When restarting, the fix and the Colvars module should be created and configured using the original configuration file(s).

2.38.5 Output

This fix computes a global scalar which can be accessed by various [output commands](#). The scalar is the energy due to all external potentials defined in the Colvars configuration. The scalar value calculated by this fix is “extensive”.

Aside from the state information in a “.colvars.state” file, other [output files](#) are produced by Colvars depending on the type of simulation. For this reason, the “output” keyword is required for fix colvars.

2.38.6 Controlling Colvars via *fix_modify*

Added in version Colvars_2023-06-04.

The *fix_modify* command may be used on “fix colvars” in either one of two ways:

- (1) Provide updated values for the fix parameters, such as *output*, *input*, *unwrap*, *tstat* and *seed*. Additionally, the *fix_modify energy* keyword is supported by this fix to add the energy change from the biasing force added by Colvars to the global potential energy of the system as part of *thermodynamic output* (the default is *fix_modify energy no*). For example, in a multi-step LAMMPS script involving multiple thermostats (e.g. fix nvt followed by fix npt), Colvars can read a new thermostat’s target temperature like this:

```
fix NVT all nvt ...
fix Colvars all colvars <configfile> output equil1 tstat NVT
run <NUMSTEPS>
unfix nvt
fix NPT all n ...
fix_modify Colvars tstat NPT
fix_modify Colvars output equil2
```

- (2) Call one of the scripting functions provided by the Colvars module itself (a full list is available in the Colvars doc). The arguments to these functions are provided as strings and passed to Colvars.

LAMMPS variables referenced by their string representation “\${variable}” will be expanded immediately. Note also that this variable expansion *will also happen within quotes*, similar to what the *mdi* command provides. This feature makes it possible to use the values of certain LAMMPS variables in Colvars configuration strings. For example, to synchronize the LAMMPS and Colvars dump frequencies:

```
variable freq index 10000
dump myDump all atom/zstd ${freq} dump.atom.zstd
fix_modify Colvars config "colvarsTrajFrequency ${freq}"
```

Note

Although it is possible to use *fix_modify* at any time, its results will only reflect the state of the Colvars module at the end of the most recent “run” or “minimize” command. Any new configuration added via “fix_modify Colvars configfile” or “fix_modify Colvars config” will only be loaded when the simulation resumes. Configuration files or strings will be parsed in the same sequence as they were provided in the LAMMPS script.

2.38.7 Restrictions

This fix is provided by the COLVARS package and is only available if LAMMPS was built with that package (default in most builds). Some of the features also require code available from the LEPTON package. See the *Build package* page for more info.

There can only be one Colvars instance defined at a time. Since the interface communicates only the minimum required amount of information, and the Colvars module itself can handle an arbitrary number of collective variables, this is not a limitation of functionality.

2.38.8 Related commands

fix smd, fix spring, fix plumed

(Fiorin) Fiorin, Klein, Henin, Mol. Phys. 111, 3345 (2013) <https://doi.org/10.1080/00268976.2013.813594>

<https://colvars.github.io/colvars-refman-lammps/colvars-refman-lammps.html>

2.39 fix controller command

2.39.1 Syntax

```
fix ID group-ID controller Nevery alpha Kp Ki Kd pvar setpoint cvar
```

- ID, group-ID are documented in [fix](#) command
- controller = style name of this fix command
- Nevery = invoke controller every this many timesteps
- alpha = coupling constant for PID equation (see units discussion below)
- Kp = proportional gain in PID equation (unitless)
- Ki = integral gain in PID equation (unitless)
- Kd = derivative gain in PID equation (unitless)
- pvar = process variable of form c_ID, c_ID[I], f_ID, f_ID[I], or 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 = value calculated by an equal-style variable with name

- setpoint = desired value of process variable (same units as process variable)
- cvar = name of control variable

2.39.2 Examples

```
fix 1 all controller 100 1.0 0.5 0.0 0.0 c_thermo_temp 1.5 tcontrol
fix 1 all controller 100 0.2 0.5 0 100.0 v_pxxwall 1.01325 xwall
fix 1 all controller 10000 0.2 0.5 0 2000 v_avpe -3.785 tcontrol
```

2.39.3 Description

This fix enables control of a LAMMPS simulation using a control loop feedback mechanism known as a proportional-integral-derivative (PID) controller. The basic idea is to define a “process variable” which is a quantity that can be monitored during a running simulation. A desired target value is chosen for the process variable. A “control variable” is also defined which is an adjustable attribute of the running simulation, which the process variable will respond to. The PID controller continuously adjusts the control variable based on the difference between the process variable and the target.

Here are examples of ways in which this fix can be used. The examples/pid directory contains a script that implements the simple thermostat.

Goal	process variable	control variable
Simple thermostat	instantaneous T	thermostat target T
Find melting temperature	average PE per atom	thermostat target T
Control pressure in non-periodic system	force on wall	position of wall

Note

For this fix to work, the control variable must actually induce a change in a running LAMMPS simulation. Typically this will only occur if there is some other command (e.g. a thermostat fix) which uses the control variable as an input parameter. This could be done directly or indirectly, e.g. the other command uses a variable as input whose formula uses the control variable. The other command should alter its behavior dynamically as the variable changes.

Note

If there is a command you think could be used in this fashion, but does not currently allow a variable as an input parameter, please notify the LAMMPS developers. It is often not difficult to enable a command to use a variable as an input parameter.

The group specified with this command is ignored. However, note that the process variable may be defined by calculations performed by computes and fixes which store their own “group” definitions.

The PID controller is invoked once each *Nevery* timesteps.

The PID controller is implemented as a discretized version of the following dynamic equation:

$$\frac{dc}{dt} = -\alpha(K_p e + K_i \int_0^t e dt + K_d \frac{de}{dt})$$

where c is the continuous time analog of the control variable, $e = pvar - setpoint$ is the error in the process variable, and α , K_p , K_i , and K_d are constants set by the corresponding keywords described above. The discretized version of this equation is:

$$c_n = c_{n-1} - \alpha \left(K_p \tau e_n + K_i \tau^2 \sum_{i=1}^n e_i + K_d (e_n - e_{n-1}) \right)$$

where $\tau = N_{every} \cdot timestep$ is the time interval between updates, and the subscripted variables indicate the values of c and e at successive updates.

From the first equation, it is clear that if the three gain values K_p , K_i , K_d are dimensionless constants, then α must have units of [unit *cvar*]/[unit *pvar*]/[unit time] e.g. [eV/K/ps]. The advantage of this unit scheme is that the value of the

constants should be invariant under a change of either the MD timestep size or the value of *Nevery*. Similarly, if the LAMMPS *unit style* is changed, it should only be necessary to change the value of α to reflect this, while leaving K_p , K_i , and K_d unaltered.

When choosing the values of the four constants, it is best to first pick a value and sign for α that is consistent with the magnitudes and signs of *pvar* and *cvar*. The magnitude of K_p should then be tested over a large positive range keeping $K_i = K_d = 0$. A good value for K_p will produce a fast response in *pvar*, without overshooting the *setpoint*. For many applications, proportional feedback is sufficient, and so $K_i = K_d = 0$ can be used. In cases where there is a substantial lag time in the response of *pvar* to a change in *cvar*, this can be counteracted by increasing K_d . In situations where *pvar* plateaus without reaching *setpoint*, this can be counteracted by increasing K_i . In the language of Charles Dickens, K_p represents the error of the present, K_i the error of the past, and K_d the error yet to come.

Because this fix updates *cvar*, but does not initialize its value, the initial value c_0 is that assigned by the user in the input script via the *internal-style variable* command. This value is used (by every other LAMMPS command that uses the variable) until this fix performs its first update of *cvar* after *Nevery* timesteps. On the first update, the value of the derivative term is set to zero, because the value of e_{n-1} is not yet defined.

The process variable *pvar* can be specified as the output of a *compute* or *fix* or the evaluation of a *variable*. In each case, the compute, fix, or variable must produce a global quantity, not a per-atom or local quantity.

If *pvar* begins with “c_”, a compute ID must follow which has been previously defined in the input script and which generates a global scalar or vector. See the individual *compute* doc page for details. If no bracketed integer is appended, the scalar calculated by the compute is used. If a bracketed integer is appended, the *I*th value of the vector calculated by the compute is used. Users can also write code for their own compute styles and *add them to LAMMPS*.

If *pvar* begins with “f_”, a fix ID must follow which has been previously defined in the input script and which generates a global scalar or vector. See the individual *fix* page for details. Note that some fixes only produce their values on certain timesteps, which must be compatible with when fix controller references the values, or else an error results. If no bracketed integer is appended, the scalar calculated by the fix is used. If a bracketed integer is appended, the *I*th value of the vector calculated by the fix is used. Users can also write code for their own fix style and *add them to LAMMPS*.

If *pvar* 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 the process variable.

The target value *setpoint* for the process variable must be a numeric value, in whatever units *pvar* is defined for.

The control variable *cvar* must be the name of an *internal-style variable* previously defined in the input script. Note that it is not specified with a “v_” prefix, just the name of the variable. It must be an internal-style variable, because this fix updates its value directly. Note that other commands can use an equal-style versus internal-style variable interchangeably.

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

Currently, 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 with 3 values which can be accessed by various *output commands*. The values can be accessed on any timestep, though they are only updated on timesteps that are a multiple of *Nevery*.

The three values are the most recent updates made to the control variable by each of the 3 terms in the PID equation above. The first value is the proportional term, the second is the integral term, the third is the derivative term.

The units of the vector values will be whatever units the control variable is in. 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*.

2.39.5 Restrictions

This fix is part of the EXTRA-FIX package. It is only enabled if LAMMPS was built with that package. See the *Build package* page for more info.

2.39.6 Related commands

fix adapt

2.39.7 Default

none

2.40 fix damping/cundall command

2.40.1 Syntax

```
fix ID group-ID damping/cundall gamma_1 gamma_a keyword values ...
```

- ID, group-ID are documented in *fix* command
- damping/cundall = style name of this fix command
- gamma_1 = linear damping coefficient (dimensionless)
- gamma_a = angular damping coefficient (dimensionless)
- zero or more keyword/value pairs may be appended

keyword = scale

scale values = type ratio or v_name

type = atom type (1-N)

ratio = factor to scale the damping coefficients by

v_name = reference to atom style variable name

2.40.2 Examples

```
fix 1 all damping/cundall 0.8 0.8
fix 1 all damping/cundall 0.8 0.5 scale 3 2.5
fix a all damping/cundall 0.8 0.5 scale v_radscale
```

2.40.3 Description

Add damping force and torque to finite-size spherical particles in the group following the model of [Cundall, 1987](#), as implemented in other granular physics code (e.g., [Yade-DEM](#), [PFC](#)).

The damping is constructed to always have negative mechanical power with respect to the current velocity/angular velocity to ensure dissipation of kinetic energy. If used without additional thermostating (to add kinetic energy to the system), it has the effect of slowly (or rapidly) freezing the system; hence it can also be used as a simple energy minimization technique.

The magnitude of the damping force/torque F_d/T_d is a fraction $\gamma \in [0; 1]$ of the current force/torque F/T on the particle. Damping is applied component-by-component in each direction $k \in \{x, y, z\}$:

$$F_{dk} = -\gamma_l F_k \operatorname{sign}(F_k v_k)$$

$$T_{dk} = -\gamma_a T_k \operatorname{sign}(T_k \omega_k)$$

The larger the coefficients, the faster the kinetic energy is reduced.

If the optional keyword `scale` is used, γ_l and γ_a can be scaled up or down by the specified factor for atoms. This factor can be set for different atom types and thus the `scale` keyword used multiple times followed by the atom type and the associated scale factor. Alternately the scaling factor can be computed for each atom (e.g. based on its radius) by using an [atom-style variable](#).

Note

The damping force/torque is computed based on the force/torque at the moment this fix is invoked. Any force/torque added after this fix, e.g., by [fix addforce](#) or [fix addtorque](#) will not be damped. When performing simulations with gravity, invoking [fix gravity](#) after this fix will maintain the specified gravitational acceleration.

Note

This scheme is dependent on the coordinates system and does not correspond to realistic physical processes. It is constructed for numerical convenience and efficacy.

This non-viscous damping presents the following advantages:

1. damping is independent of velocity, equally damping regions with distinct natural frequencies,
2. damping affects acceleration and vanishes for steady uniform motion of the particles,
3. damping parameter γ is dimensionless and does not require scaling.

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

No information about this fix is written to [binary restart files](#). 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 [fix_modify respa](#) option is supported by this fix. This allows to set at which level of the [r-RESPA](#) integrator the fix is modifying forces/torques. Default is the outermost level.

The forces/torques 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.

2.40.5 Restrictions

This fix is part of the GRANULAR package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

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

2.40.6 Related commands

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

2.40.7 Default

none

2.40.8 References

(Cundall, 1987) Cundall, P. A. Distinct Element Models of Rock and Soil Structure, in Analytical and Computational Methods in Engineering Rock Mechanics, Ch. 4, pp. 129-163. E. T. Brown, ed. London: Allen & Unwin., 1987.

(PFC) PFC Particle Flow Code 6.0 Documentation. Itasca Consulting Group.

(Yade-DEM) V. Smilauer et al. (2021), Yade Documentation 3rd ed. The Yade Project. DOI:10.5281/zenodo.5705394 (<https://yade-dem.org/doc/>)

2.41 fix deform command

Accelerator Variants: *deform/kk*

2.41.1 Syntax

```
fix ID group-ID deform N parameter style args ... keyword value ...
```

- ID, group-ID are documented in [fix](#) command
- N = perform box deformation every this many timesteps
- one or more parameter/args sequences 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 variable or pressure
or pressure/mean
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 or variable
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* or *couple* or *vol/balance/p* or *max/rate* or *normalize/pressure*

remap value = x or v or none
x = remap coords of atoms in group into deforming box
v = remap velocities of atoms in group 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

2.41.2 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
```

2.41.3 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.

The [fix deform/pressure](#) command extends this command with additional keywords and arguments. The rest of this page explains the options common to both commands. The [fix deform/pressure](#) page explains the options available ONLY with the fix deform/pressure command. Note that a simulation can define only a single deformation command: fix deform or fix deform/pressure.

Both these fixes can be used to perform non-equilibrium MD (NEMD) simulations of a continuously strained system. See the [fix nvt/sllod](#) and [compute temp/deform](#) commands for more details. Note that simulation of a continuously extended system (extensional flow) can be modeled using the [UEF package](#) and its [fix commands](#).

Inconsistent trajectories due to image flags

When running long simulations while shearing the box or using a high shearing rate, it is possible that the image flags used for storing unwrapped atom positions will “wrap around”. When LAMMPS is compiled with the default settings, case image flags are limited to a range of $-512 \leq i \leq 511$, which will overflow when atoms starting at zero image flag value have passed through a periodic box dimension more than 512 times.

Changing the [size of LAMMPS integer types](#) to the “bigbig” setting can make this overflow much less likely, since it increases the image flag value range to $-1,048,576 \leq i \leq 1,048,575$

For the x , y , z parameters, the associated dimension cannot be shrink-wrapped. For the xy , yz , xz parameters, the associated second 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 $\text{rate} = V/L_0$ and L_0 is the initial box length. The distance can be in lattice or box distance units. See the discussion of the units keyword below. For example, if the initial box length is 100 Angstroms, and V is 10 Angstroms/ps, then after 10 ps, the box length will have doubled. After 20 ps, it will have tripled.

The *erate* style changes a dimension of 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 Δ/L_0 , where L_0 is the original box length and Δ is the change relative to the original length. The box length L as a function of time will change as

$$L(t) = L_0 (1 + \text{erate} * dt)$$

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 ps = 0.1, strain after 2 ps = 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 than 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 Δ/L_0 , where L_0 is the original box length and Δ is the change relative to the original length.

The box length L as a function of time will change as

$$L(t) = L_0 \exp(\text{trate} * dt)$$

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 ps = 0.1, strain after 2 ps = 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 $M = \text{pow}(2,1/3) - 1 = 0.26$, since if each box dimension grows by 26%, the box volume doubles. Or you could set “x trate M”, “y trate M”, “z trate M”, with $M = \ln(1.26) = 0.231$, and the box volume would double every picosecond.

The *volume* style changes the specified dimension in such a way that the box volume remains constant while other box dimensions are changed explicitly via the styles discussed above. For example, “x scale 1.1 y scale 1.1 z volume” will shrink the z box length as the x,y box lengths increase, to keep the volume constant (product of x,y,z lengths). If “x scale 1.1 z volume” is specified and parameter y is unspecified, then the z box length will shrink as x increases to keep the product of x,z lengths constant. If “x scale 1.1 y volume z volume” is specified, then both the y,z box lengths will shrink as x increases to keep the volume constant (product of x,y,z lengths). In this case, the y,z box lengths shrink so as to keep their relative aspect ratio constant.

For solids or liquids, note that when one dimension of the box is expanded via fix deform (i.e. tensile strain), it may be physically undesirable to hold the other 2 box lengths constant (unspecified by fix deform) since that implies a density change. Using the *volume* style for those 2 dimensions to keep the box volume constant may make more physical sense, but may also not be correct for materials and potentials whose Poisson ratio is not 0.5. An alternative is to use *fix npt aniso* with zero applied pressure on those 2 dimensions, so that they respond to the tensile strain dynamically.

The *wiggle* style oscillates the specified box length dimension sinusoidally with the specified amplitude and period. I.e. the box length L as a function of time is given by

$$L(t) = L_0 + A \sin(2\pi t/T_p)$$

where L_0 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 LAMMPS 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 $\text{rate} = V/L_0$ and L_0 is the initial box length perpendicular to the direction of shear. The distance can be in lattice or box distance units. See the discussion of the *units* keyword below. For example, if the initial tilt factor is 5 Angstroms, and the V is 10 Angstroms/ps, then after 1 ps, the tilt factor will be 15 Angstroms. After 2 ps, 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

$$T(t) = T_0 + L_0 * \text{erate} * dt$$

where T_0 is the initial tilt factor, L_0 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 ps = 0.1, strain after 2 ps = 0.2, etc. Thus the tilt factor would be 0.0 at time 0, $0.1 * y_{\text{box}}$ at 1 ps, $0.2 * y_{\text{box}}$ at 2 ps, etc, where y_{box} is the original y box length. $R = 1$ or 2 means the tilt factor will increase by 1 or 2 every picosecond. $R = -0.01$ means a decrease in shear strain by 0.01 every picosecond.

The *trate* style changes a tilt factor at a “constant true shear strain rate”. Note that this is not an “engineering shear strain rate”, as the other styles are. Rather, for a “true” rate, the rate of change is constant, which means the tilt factor changes non-linearly with time from its initial to final value. The units of the specified shear strain rate are 1/time. See the [units](#) command for the time units associated with different choices of simulation units, e.g. picoseconds for “metal” units). Shear strain is unitless and is defined as offset/length, where length is the box length perpendicular to the shear direction (e.g. y box length for xy deformation) and offset is the displacement distance in the shear direction (e.g. x direction for xy deformation) from the unstrained orientation.

The tilt factor T as a function of time will change as

$$T(t) = T_0 \exp(\text{trate} * dt)$$

where T_0 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 ps = 0.11, strain after 2 ps = 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

$$T(t) = T_0 + A \sin(2 * \pi * t / T_p)$$

where T_0 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 LAMMPS 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 LAMMPS, 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 first 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.

Note

When non-equilibrium MD (NEMD) simulations are performed using this fix, the option “*remap v*” should normally be used. This is because *fix nvt/sllod* adjusts the atom positions and velocities to induce a velocity profile that matches the changing box size/shape. Thus atom coordinates should NOT be remapped by *fix deform*, but

velocities SHOULD be when atoms cross periodic boundaries, since that is consistent with maintaining the velocity profile already created by fix nvt/sllod. LAMMPS will warn you if the *remap* setting is not consistent with fix nvt/sllod.

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 langevin](#) (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/chunk](#), [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.

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 LAMMPS will run less efficiently, due to the large volume of communication needed to acquire ghost atoms around a processor’s irregular-shaped subdomain. For extreme values of tilt, LAMMPS 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.

Styles with a *gpu*, *intel*, *kk*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed on the [Accelerator packages](#) page. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the GPU, INTEL, KOKKOS, OPENMP, and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the [Build package](#) page for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the [-suffix command-line switch](#) when you invoke LAMMPS, or you can use the [suffix](#) command in your input script.

See the [Accelerator packages](#) page for more instructions on how to use the accelerated styles effectively.

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

This fix will restore the initial box settings from [binary restart files](#), which allows the fix to be properly continue deformation, when using the start/stop options of the [run](#) command. 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*.

2.41.5 Restrictions

You cannot apply x, y, or z deformations to a dimension that is shrink-wrapped via the [boundary](#) command.

You cannot apply xy, yz, or xz deformations to a second dimension (y in xy) that is shrink-wrapped via the [boundary](#) command.

2.41.6 Related commands

[fix deform/pressure](#), [change_box](#)

2.41.7 Default

The option defaults are remap = x, flip = yes, and units = lattice.

2.42 fix deform/pressure command

2.42.1 Syntax

```
fix ID group-ID deform/pressure N parameter style args ... keyword value ...
```

- ID, group-ID are documented in [fix](#) command
 - deform/pressure = style name of this fix command
 - N = perform box deformation every this many timesteps
 - one or more parameter/args sequences may be appended
- parameter = x or y or z or xy or xz or yz or box
x, y, z args = style value(s)
style = final or delta or scale or vel or erate or trate or volume or wiggle or variable or pressure
→ or pressure/mean
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
pressure values = target gain
target = target pressure (pressure units)
gain = proportional gain constant (1/(time * pressure) or 1/time units)
pressure/mean values = target gain
target = target pressure (pressure units)
gain = proportional gain constant (1/(time * pressure) or 1/time units)

xy, xz, yz args = style value
style = final or delta or vel or erate or trate or wiggle or variable or pressure or erate/rescale
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)
erate/rescale 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
pressure values = target gain
target = target pressure (pressure units)
gain = proportional gain constant (1/(time * pressure) or 1/time units)
erate/rescale value = R
R = engineering shear strain rate (1/time units)

box = style value
style = volume or pressure
volume value = none = isotropically adjust system to preserve volume of system
pressure values = target gain
target = target mean pressure (pressure units)
gain = proportional gain constant (1/(time * pressure) or 1/time units)

- zero or more keyword/value pairs may be appended
- keyword = *remap* or *flip* or *units* or *couple* or *vol/balance/p* or *max/rate* or *normalize/pressure*

remap value = x or v or none

 x = remap coords of atoms in group into deforming box

 v = remap velocities of atoms in group 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

couple value = none or xyz or xy or yz or xz

 couple pressure values of various dimensions

vol/balance/p value = yes or no

 Modifies the behavior of the volume option to try and balance pressures

max/rate value = rate

 rate = maximum strain rate for pressure control

normalize/pressure value = yes or no

 Modifies pressure controls such that the deviation in pressure is normalized by the target pressure

2.42.2 Examples

```
fix 1 all deform/pressure 1 x pressure 2.0 0.1 normalize/pressure yes max/rate 0.001
fix 1 all deform/pressure 1 x trate 0.1 y volume z volume vol/balance/p yes
fix 1 all deform/pressure 1 x trate 0.1 y pressure/mean 0.0 1.0 z pressure/mean 0.0 1.0
```

2.42.3 Description

Added in version 17Apr2024.

This fix is an extension of the [fix deform](#) command, which allows all of its options to be used as well as new pressure-based controls implemented by this command.

All arguments described on the [fix deform](#) doc page also apply to this fix unless otherwise noted below. The rest of this page explains the arguments specific to this fix only. Note that a simulation can define only a single deformation command: fix deform or fix deform/pressure.

Inconsistent trajectories due to image flags

When running long simulations while shearing the box or using a high shearing rate, it is possible that the image flags used for storing unwrapped atom positions will “wrap around”. When LAMMPS is compiled with the default settings, case image flags are limited to a range of $-512 \leq i \leq 511$, which will overflow when atoms starting at zero image flag value have passed through a periodic box dimension more than 512 times.

Changing the [size of LAMMPS integer types](#) to the “bigbig” setting can make this overflow much less likely, since it increases the image flag value range to $-1,048,576 \leq i \leq 1,048,575$

For the x, y, and z parameters, this is the meaning of the styles and values provided by this fix.

The *pressure* style adjusts a dimension's box length to control the corresponding component of the pressure tensor. This option attempts to maintain a specified target pressure using a linear controller where the box length L evolves according to the equation

$$\frac{dL(t)}{dt} = L(t)k(P_t - P)$$

where k is a proportional gain constant, P_t is the target pressure, and P is the current pressure along that dimension. This approach is similar to the method used to control the pressure by [fix press/berendsen](#). The target pressure accepts either a constant numeric value or a LAMMPS *variable*. Notably, this variable can be a function of time or other components of the pressure tensor. By default, k has units of $1/(time * pressure)$ although this will change if the *normalize/pressure* option is set as [discussed below](#). There is no proven method to choosing an appropriate value of k as it will depend on the specific details of a simulation. Testing different values is recommended.

By default, there is no limit on the resulting strain rate in any dimension. A maximum limit can be applied using the *max/rate* option. Akin to [fix npt and nph](#), pressures in different dimensions can be coupled using the *couple* option. This means the instantaneous pressure along coupled dimensions are averaged and the box strains identically along the coupled dimensions.

The *pressure/mean* style changes a dimension's box length to maintain a constant mean pressure defined as the trace of the pressure tensor. This option has identical arguments to the *pressure* style and a similar functional equation, except the current and target pressures refer to the mean trace of the pressure tensor. All options for the *pressure* style also apply to the *pressure/mean* style except for the *couple* option.

Note that while this style can be identical to coupled *pressure* styles, it is generally not the same. For instance in 2D, a coupled *pressure* style in the x and y dimensions would be equivalent to using the *pressure/mean* style with identical settings in each dimension. However, it would not be the same if settings (e.g. gain constants) were used in the x and y dimensions or if the *pressure/mean* command was only applied along one dimension.

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

The *pressure* style adjusts a tilt factor to control the corresponding off-diagonal component of the pressure tensor. This option attempts to maintain a specified target value using a linear controller where the tilt factor T evolves according to the equation

$$\frac{dT(t)}{dt} = L(t)k(P - P_t)$$

where k is a proportional gain constant, P_t is the target pressure, P is the current pressure, and L is the perpendicular box length. The target pressure accepts either a constant numeric value or a LAMMPS *variable*. Notably, this variable can be a function of time or other components of the pressure tensor. By default, k has units of $1/(time * pressure)$ although this will change if the *normalize/pressure* option is set as [discussed below](#). There is no proven method to choosing an appropriate value of k as it will depend on the specific details of a simulation and testing different values is recommended. One can also apply a maximum limit to the magnitude of the applied strain using the *max/rate* option.

The *erate/rescale* style operates similarly to the *erate* style with a specified strain rate in units of $1/time$. The difference is that the change in the tilt factor will depend on the current length of the box perpendicular to the shear direction, L , instead of the original length, $L0$. The tilt factor T as a function of time will change as

$$T(t) = T(t - 1) + L \cdot erate \cdot \Delta t$$

where $T(t-1)$ is the tilt factor on the previous timestep and Δt is the timestep size. This option may be useful in scenarios where L changes in time.

The *box* parameter provides an additional control over the x , y , and z box lengths by isotropically dilating or contracting the box to either maintain a fixed mean pressure or volume. This isotropic scaling is applied after the box is deformed

by the above x , y , z , xy , xz , and yz styles, acting as a second deformation step. This parameter will change the overall strain rate in the x , y , or z dimensions. This parameter can only be used in combination with the x , y , or z commands: *vel*, *erate*, *trate*, *pressure*, or *wiggle*. This is the meaning of its styles and values.

The *volume* style isotropically scales box lengths to maintain a constant box volume in response to deformation from other parameters. This style may be useful in scenarios where one wants to apply a constant deviatoric pressure using *pressure* styles in the x , y , and z dimensions (deforming the shape of the box), while maintaining a constant volume.

The *pressure* style isotropically scales box lengths in an attempt to maintain a target mean pressure (the trace of the pressure tensor) of the system. This is accomplished by isotropically scaling all box lengths L by an additional factor of $k(P_t - P_m)$ where k is the proportional gain constant, P_t is the target pressure, and P_m is the current mean pressure. This style may be useful in scenarios where one wants to apply a constant deviatoric strain rate using various strain-based styles (e.g. *trate*) along the x , y , and z dimensions (deforming the shape of the box), while maintaining a mean pressure.

The optional keywords provided by this fix are described below.

The *normalize/pressure* keyword changes how box dimensions evolve when using the *pressure* or *pressure/mean* deformation styles. If the *deform/normalize* value is set to *yes*, then the deviation from the target pressure is normalized by the absolute value of the target pressure such that the proportional gain constant scales a percentage error and has units of 1/time. If the target pressure is ever zero, this will produce an error unless the *max/rate* keyword is defined, described below, which will cap the divergence.

The *max/rate* keyword sets an upper threshold, *rate*, that limits the maximum magnitude of the instantaneous strain rate applied in any dimension. This keyword only applies to the *pressure* and *pressure/mean* options. If a pressure-controlled rate is used for both *box* and either x , y , or z , then this threshold will apply separately to each individual controller such that the cumulative strain rate on a box dimension may be up to twice the value of *rate*.

The *couple* keyword allows two or three of the diagonal components of the pressure tensor to be “coupled” together for the *pressure* option. The value specified with the keyword determines which are coupled. For example, *xz* means the P_{xx} and P_{zz} 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. If a *pressure* style is defined for more than one coupled dimension, the target pressures and gain constants must be identical. Alternatively, if a *pressure* style is only defined for one of the coupled dimensions, its settings are copied to other dimensions with undefined styles. *Couple xyz* can be used for a 2d simulation; the z dimension is simply ignored.

The *vol/balance/p* keyword modifies the behavior of the *volume* style when applied to two of the x , y , and z dimensions. Instead of straining the two dimensions in lockstep, the two dimensions are allowed to separately dilate or contract in a manner to maintain a constant volume while simultaneously trying to keep the pressure along each dimension equal using a method described in ([Huang2014](#)).

If any pressure controls are used, this fix computes a temperature and pressure each timestep. To do this, the fix creates its own computes of style “temp” and “pressure”, as if these commands had been issued:

```
compute fix-ID_temp group-ID temp
compute fix-ID_press group-ID pressure fix-ID_temp
```

See the *compute temp* and *compute pressure* commands for details. Note that the IDs of the new computes are the fix-ID + underscore + “temp” or fix_ID + underscore + “press”, and the group for the new computes is the same as the fix group.

Note that these are NOT the computes used by thermodynamic output (see the *thermo_style* command) with ID = *thermo_temp* and *thermo_press*. This means you can change the attributes of this fix’s temperature or pressure via the *compute_modify* command or print this temperature or pressure during thermodynamic output via the *thermo_style*

custom command using the appropriate compute-ID. It also means that changing attributes of *thermo_temp* or *thermo_press* will have no effect on this fix.

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

This fix will restore the initial box settings from *binary restart files*, which allows the fix to be properly continue deformation, when using the start/stop options of the *run* command. No global or per-atom quantities are stored by this fix for access by various *output commands*.

If any pressure controls are used, the *fix_modify temp* and *press* options are supported by this fix, unlike in *fix deform*. You can use them to assign a *compute* you have defined to this fix which will be used in its temperature and pressure calculations. If you do this, note that the kinetic energy derived from the compute temperature should be consistent with the virial term computed using all atoms for the pressure. LAMMPS will warn you if you choose to compute temperature on a subset of atoms.

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*.

2.42.5 Restrictions

This fix is part of the EXTRA-FIX package. It is only enabled if LAMMPS was built with that package. See the *Build package* page for more info.

You cannot apply x, y, or z deformations to a dimension that is shrink-wrapped via the *boundary* command.

You cannot apply xy, yz, or xz deformations to a second dimension (y in xy) that is shrink-wrapped via the *boundary* command.

2.42.6 Related commands

fix deform, change_box

2.42.7 Default

The option defaults are normalize/pressure = no.

(Huang2014) X. Huang, “Exploring critical-state behavior using DEM”, Doctoral dissertation, Imperial College. (2014). <https://doi.org/10.25560/25316>

2.43 fix deposit command

2.43.1 Syntax

```
fix ID group-ID deposit N type M seed keyword values ...
```

- ID, group-ID are documented in *fix* command
- deposit = style name of this fix command

- N = # of atoms or molecules to insert
- type = atom type (1-Ntypes or type label) to assign to inserted atoms (offset for molecule insertion)
- M = insert a single atom or molecule every M steps
- seed = random # seed (positive integer)
- one or more keyword/value pairs may be appended to args
- keyword = *region* or *var* or *set* or *id* or *global* or *local* or *near* or *gaussian* or *attempt* or *rate* or *vx* or *vy* or *vz* or *target* or *mol* or *molfrac* or *rigid* or *shake* or *orient* or *units*

region value = region-ID

region-ID = ID of region to use as insertion volume

var value = name = variable name to evaluate for test of atom creation

set values = dim name

dim = x or y or z

name = name of variable to set with x, y, or z atom position

id value = max or next

max = atom ID for new atom(s) is max ID of all current atoms plus one

next = atom ID for new atom(s) increments by one for every deposition

global values = lo hi

lo,hi = put new atom/molecule a distance lo-hi above all other atoms (distance units)

local values = lo hi delta

lo,hi = put new atom/molecule a distance lo-hi above any nearby atom beneath it (distance units)

delta = lateral distance within which a neighbor is considered "nearby" (distance units)

near value = R

R = only insert atom/molecule if further than R from existing particles (distance units)

gaussian values = xmid ymid zmid sigma

xmid,ymid,zmid = center of the gaussian distribution (distance units)

sigma = width of gaussian distribution (distance units)

attempt value = Q

Q = attempt a single insertion up to Q times

rate value = V

V = z velocity (y in 2d) at which insertion volume moves (velocity units)

vx values = vxlo vxhi

vxlo,vxhi = range of x velocities for inserted atom/molecule (velocity units)

vy values = vylo vyhi

vylo,vyhi = range of y velocities for inserted atom/molecule (velocity units)

vz values = vzlo vzhi

vzlo,vzhi = range of z velocities for inserted atom/molecule (velocity units)

target values = tx ty tz

tx,ty,tz = location of target point (distance units)

mol value = template-ID

template-ID = ID of molecule template specified in a separate [molecule](#) command
molfrac values = f1 f2 ... fn

f1 to fn = relative probability of creating each of N molecules in template-ID
rigid value = fix-ID

fix-ID = ID of [fix rigid/small](#) command

shake value = fix-ID

fix-ID = ID of [fix shake](#) command

orient values = rx ry rz

rx,ry,rz = vector to randomly rotate an inserted molecule around

units value = lattice or box

lattice = the geometry is defined in lattice units

box = the geometry is defined in simulation box units

2.43.2 Examples

```

fix 3 all deposit 1000 2 100 29494 region myblock local 1.0 1.0 1.0 units box
fix 2 newatoms deposit 10000 1 500 12345 region disk near 2.0 vz -1.0 -0.8
fix 4 sputter deposit 1000 2 500 12235 region sphere vz -1.0 -1.0 target 5.0 5.0 0.0 units lattice
fix 5 insert deposit 200 2 100 777 region disk gaussian 5.0 5.0 9.0 1.0 units box

labelmap atom 1 Au
fix 4 sputter deposit 1000 Au 500 12235 region sphere vz -1.0 -1.0 target 5.0 5.0 0.0 units lattice

```

2.43.3 Description

Insert a single atom or molecule into the simulation domain every M timesteps until N atoms or molecules have been inserted. This is useful for simulating deposition onto a surface. For the remainder of this doc page, a single inserted atom or molecule is referred to as a “particle”.

If inserted particles are individual atoms, they are assigned the specified atom type. If they are molecules, the type of each atom in the inserted molecule is specified in the file read by the [molecule](#) command, and those values are added to the specified atom type. E.g. if the file specifies atom types 1,2,3, and those are the atom types you want for inserted molecules, then specify *type* = 0. If you specify *type* = 2, the in the inserted molecule will have atom types 3,4,5.

All atoms in the inserted particle are assigned to two groups: the default group “all” and the group specified in the fix deposit command (which can also be “all”).

If you are computing temperature values which include inserted particles, you will want to use the [compute_modify dynamic/dof yes](#) option, which ensures the current number of atoms is used as a normalizing factor each time the temperature is computed.

Care must be taken that inserted particles are not too near existing atoms, using the options described below. When inserting particles above a surface in a non-periodic box (see the [boundary](#) command), the possibility of a particle escaping the surface and flying upward should be considered, since the particle may be lost or the box size may grow infinitely large. A [fix wall/reflect](#) command can be used to prevent this behavior. Note that if a shrink-wrap boundary is used, it is OK to insert the new particle outside the box, however the box will immediately be expanded to include the new particle. When simulating a sputtering experiment it is probably more realistic to ignore those atoms using the [thermo_modify](#) command with the *lost ignore* option and a fixed [boundary](#).

The fix deposit command must use the *region* keyword to define an insertion volume. The specified region must have been previously defined with a [region](#) command. It must be defined with side = *in*.

Note

LAMMPS checks that the specified region is wholly inside the simulation box. It can do this correctly for orthonormal simulation boxes. However for *triclinic boxes*, it only tests against the larger orthonormal box that bounds the tilted simulation box. If the specified region includes volume outside the tilted box, then an insertion will likely fail, leading to a “lost atoms” error. Thus for triclinic boxes you should ensure the specified region is wholly inside the simulation box.

The locations of inserted particles are taken from uniform distributed random numbers, unless the *gaussian* keyword is used. Then the individual coordinates are taken from a gaussian distribution of width *sigma* centered on *xmid,ymid,zmid*.

Individual atoms are inserted, unless the *mol* keyword is used. It specifies a *template-ID* previously defined using the [molecule](#) command, which reads files that define one or more molecules. The coordinates, atom types, charges, etc, as well as any bond/angle/etc and special neighbor information for the molecule can be specified in the molecule file. See

the [molecule](#) command for details. The only settings required to be in each file are the coordinates and types of atoms in the molecule.

If the molecule template contains more than one molecule, the relative probability of depositing each molecule can be specified by the [molfrac](#) keyword. N relative probabilities, each from 0.0 to 1.0, are specified, where N is the number of molecules in the template. Each time a molecule is deposited, a random number is used to sample from the list of relative probabilities. The N values must sum to 1.0.

If you wish to insert molecules via the [mol](#) keyword, that will be treated as rigid bodies, use the [rigid](#) keyword, specifying as its value the ID of a separate [fix rigid/small](#) command which also appears in your input script.

Note

If you wish the new rigid molecules (and other rigid molecules) to be thermostatted correctly via [fix rigid/small/nvt](#) or [fix rigid/small/npt](#), then you need to use the [fix_modify dynamic/dof yes](#) command for the rigid fix. This is to inform that fix that the molecule count will vary dynamically.

If you wish to insert molecules via the [mol](#) keyword, that will have their bonds or angles constrained via SHAKE, use the [shake](#) keyword, specifying as its value the ID of a separate [fix shake](#) command which also appears in your input script.

Each timestep a particle is inserted, the coordinates for its atoms are chosen as follows. For insertion of individual atoms, the “position” referred to in the following description is the coordinate of the atom. For insertion of molecule, the “position” is the geometric center of the molecule; see the [molecule](#) doc page for details. A random rotation of the molecule around its center point is performed, which determines the coordinates all the individual atoms.

A random position within the region insertion volume is generated. If neither the [global](#) or [local](#) keyword is used, the random position is the trial position. If the [global](#) keyword is used, the random x,y values are used, but the z position of the new particle is set above the highest current atom in the simulation by a distance randomly chosen between lo/hi. (For a 2d simulation, this is done for the y position.) If the [local](#) keyword is used, the z position is set a distance between lo/hi above the highest current atom in the simulation that is “nearby” the chosen x,y position. In this context, “nearby” means the lateral distance (in x,y) between the new and old particles is less than the [delta](#) setting.

Once a trial x,y,z position has been selected, the insertion is only performed if both the [near](#) and [var](#) keywords are satisfied (see below). If either the [near](#) or the [var](#) keyword is not satisfied, a new random position within the insertion volume is chosen and another trial is made. Up to Q attempts are made. If one or more particle insertions are not successful, LAMMPS prints a warning message.

The [near](#) keyword ensures that no current atom in the simulation is within a distance R of any atom in the new particle, including the effect of periodic boundary conditions if applicable. Note that the default value for R is 0.0, which will allow atoms to strongly overlap if you are inserting where other atoms are present. This distance test is performed independently for each atom in an inserted molecule, based on the randomly rotated configuration of the molecule.

Note

If you are inserting finite size particles or a molecule or rigid body consisting of finite-size particles, then you should typically set R larger than the distance at which any inserted particle may overlap with either a previously inserted particle or an existing particle. LAMMPS will issue a warning if R is smaller than this value, based on the radii of existing and inserted particles.

Added in version 21Nov2023.

The [var](#) and [set](#) keywords can be used together to provide a criterion for accepting or rejecting the addition of an individual atom, based on its coordinates. The [name](#) specified for the [var](#) keyword is the name of an [equal-style variable](#) that should evaluate to a zero or non-zero value based on one or two or three variables that will store the x, y,

or z coordinates of an atom (one variable per coordinate). If used, these other variables must be *internal-style variables* defined in the input script; their initial numeric value can be anything. They must be internal-style variables, because this command resets their values directly. The *set* keyword is used to identify the names of these other variables, one variable for the x -coordinate of a created atom, one for y , and one for z . When an atom is created, its (x, y, z) coordinates become the values for any *set* variable that is defined. The *var* variable is then evaluated. If the returned value is 0.0, the atom is not created. If it is non-zero, the atom is created. For an example of how to use these keywords, see the [create_atoms](#) command.

The *rate* option moves the insertion volume in the z direction (3d) or y direction (2d). This enables particles to be inserted from a successively higher height over time. Note that this parameter is ignored if the *global* or *local* keywords are used, since those options choose a z -coordinate for insertion independently.

The vx , vy , and vz components of velocity for the inserted particle are set by sampling a uniform distribution between the bounds set by the values specified for the vx , vy , and vz keywords. Note that normally, new particles should be assigned a negative vertical velocity so that they move towards the surface. For molecules, the same velocity is given to every particle (no rotation or bond vibration).

If the *target* option is used, the velocity vector of the inserted particle is changed so that it points from the insertion position towards the specified target point. The magnitude of the velocity is unchanged. This can be useful, for example, for simulating a sputtering process. E.g. the target point can be far away, so that all incident particles strike the surface as if they are in an incident beam of particles at a prescribed angle.

The *orient* keyword is only used when molecules are deposited. By default, each molecule is inserted at a random orientation. If this keyword is specified, then (rx, ry, rz) is used as an orientation vector, and each inserted molecule is rotated around that vector with a random value from zero to 2π . For a 2d simulation, $rx = ry = 0.0$ is required, since rotations can only be performed around the z axis.

The *id* keyword determines how atom IDs and molecule IDs are assigned to newly deposited particles. Molecule IDs are only assigned if molecules are being inserted. For the *max* setting, the atom and molecule IDs of all current atoms are checked. Atoms in the new particle are assigned IDs starting with the current maximum plus one. If a molecule is inserted it is assigned an ID = current maximum plus one. This means that if particles leave the system, the new IDs may replace the lost ones. For the *next* setting, the maximum ID of any atom and molecule is stored at the time the fix is defined. Each time a new particle is added, this value is incremented to assign IDs to the new atom(s) or molecule. Thus atom and molecule IDs for deposited particles will be consecutive even if particles leave the system over time.

The *units* keyword determines the meaning of the distance units used for the other deposition parameters. A *box* value selects standard distance units as defined by the [units](#) command, e.g. Angstroms for units = real or metal. A *lattice* value means the distance units are in lattice spacings. The [lattice](#) command must have been previously used to define the lattice spacing. Note that the units choice affects all the keyword values that have units of distance or velocity.

Note

If you are monitoring the temperature of a system where the atom count is changing due to adding particles, you typically should use the [compute_modify dynamic/dof yes](#) command for the temperature compute you are using.

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

This fix writes the state of the deposition to *binary restart files*. This includes information about how many particles have been deposited, the random number generator seed, the next timestep for deposition, etc. See the [read_restart](#) command for info on how to re-specify a fix in an input script that reads a restart file, so that the operation of the fix continues in an uninterrupted fashion.

Note

For this to work correctly, the timestep must **not** be changed after reading the restart with `reset_timestep`. The fix will try to detect it and stop with an error.

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 cumulative number of insertions. 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`.

2.43.5 Restrictions

The specified insertion region cannot be a “dynamic” region, as defined by the `region` command.

2.43.6 Related commands

`fix pour`, `region`

2.43.7 Default

Insertions are performed for individual atoms, i.e. no `mol` setting is defined. If the `mol` keyword is used, the default for `molfrac` is an equal probabilities for all molecules in the template. Additional option defaults are `id = max`, `delta = 0.0`, `near = 0.0`, `attempt = 10`, `rate = 0.0`, `vx = 0.0 0.0`, `vy = 0.0 0.0`, `vz = 0.0 0.0`, and `units = lattice`.

2.44 fix dpd/energy command

Accelerator Variants: `dpd/energy/kk`

2.44.1 Syntax

```
fix ID group-ID dpd/energy
```

- ID, group-ID are documented in `fix` command
- `dpd/energy` = style name of this fix command

2.44.2 Examples

```
fix 1 all dpd/energy
```

2.44.3 Description

Perform constant energy dissipative particle dynamics (DPD-E) integration. This fix updates the internal energies for particles in the group at each timestep. It must be used in conjunction with a deterministic integrator (e.g. [fix nve](#)) that updates the particle positions and velocities.

For fix *dpd/energy*, the particle internal temperature is related to the particle internal energy through a mesoparticle equation of state. An additional fix must be specified that defines the equation of state for each particle, e.g. [fix eos/cv](#).

This fix must be used with the [pair_style dpd/fdt/energy](#) command.

Note that numerous variants of DPD can be specified by choosing an appropriate combination of the integrator and [pair_style dpd/fdt/energy](#) command. DPD under isoenergetic conditions can be specified by using fix *dpd/energy*, fix *nve* and pair_style *dpd/fdt/energy*. DPD under isoenthalpic conditions can be specified by using fix *dpd/energy*, fix *nph* and pair_style *dpd/fdt/energy*. Examples of each DPD variant are provided in the examples/PACKAGES/dpd-react directory.

Styles with a *gpu*, *intel*, *kk*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed on the [Accelerator packages](#) page. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the GPU, INTEL, KOKKOS, OPENMP, and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the [Build package](#) page for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the [-suffix command-line switch](#) when you invoke LAMMPS, or you can use the [suffix](#) command in your input script.

See the [Accelerator packages](#) page for more instructions on how to use the accelerated styles effectively.

2.44.4 Restrictions

This command is part of the DPD-REACT package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

This fix must be used with an additional fix that specifies time integration, e.g. [fix nve](#).

The fix *dpd/energy* requires the *dpd atom_style* to be used in order to properly account for the particle internal energies and temperature.

The fix *dpd/energy* must be used with an additional fix that specifies the mesoparticle equation of state for each particle.

2.44.5 Related commands

[fix nve](#) [fix eos/cv](#)

2.44.6 Default

none

(Lisal) M. Lisal, J.K. Brennan, J. Bonet Avalos, J. Chem. Phys., 135, 204105 (2011).

(Larentzos) J.P. Larentzos, J.K. Brennan, J.D. Moore, and W.D. Mattson, ARL-TR-6863, U.S. Army Research Laboratory, Aberdeen Proving Ground, MD (2014).

2.45 fix edpd/source command

2.46 fix tdpd/source command

2.46.1 Syntax

```
fix ID group-ID edpd/source keyword values ...
fix ID group-ID tdpd/source cc_index keyword values ...
```

- ID, group-ID are documented in [fix](#) command
- edpd/source or tdpd/source = style name of this fix command
- index (only specified for tdpd/source) = index of chemical species (1 to Nspecies)
- keyword = *sphere* or *cuboid* or *region*

sphere args = cx cy cz radius source
 cx,cy,cz = x,y,z center of spherical domain (distance units)
 radius = radius of a spherical domain (distance units)
 source = heat source or concentration source (flux units, see below)
 cuboid values = cx cy cz dLx dLy dLz source
 cx,cy,cz = x,y,z center of a cuboid domain (distance units)
 dLx,dLy,dLz = x,y,z side length of a cuboid domain (distance units)
 source = heat source or concentration source (flux units, see below)
 region values = region-ID source
 region = ID of region for heat or concentration source
 source = heat source or concentration source (flux units, see below)

2.46.2 Examples

```
fix 1 all edpd/source sphere 0.0 0.0 0.0 5.0 0.01
fix 1 all edpd/source cuboid 0.0 0.0 0.0 20.0 10.0 10.0 -0.01
fix 1 all tdpd/source 1 sphere 5.0 0.0 0.0 5.0 0.01
fix 1 all tdpd/source 2 cuboid 0.0 0.0 0.0 20.0 10.0 10.0 0.01
fix 1 all tdpd/source 1 region lower -0.01
```

2.46.3 Description

Fix *edpd/source* adds a heat source as an external heat flux to each atom in a spherical or cuboid domain, where the *source* is in units of energy/time. Fix *tdpd/source* adds an external concentration source of the chemical species specified by *index* as an external concentration flux for each atom in a spherical or cuboid domain, where the *source* is in units of mole/volume/time.

This command can be used to give an additional heat/concentration source term to atoms in a simulation, such as for a simulation of a heat conduction with a source term (see Fig.12 in ([Li2014](#))) or diffusion with a source term (see Fig.1 in ([Li2015](#))), as an analog of a periodic Poiseuille flow problem.

Deprecated since version 15Jun2023: The *sphere* and *cuboid* keywords will be removed in a future version of LAMMPS. The same functionality and more can be achieved with a region.

If the *sphere* keyword is used, the *cx*, *cy*, *cz*, *radius* values define a spherical domain to apply the source flux to.

If the *cuboid* keyword is used, the *cx*, *cy*, *cz*, *dLx*, *dLy*, *dLz* define a cuboid domain to apply the source flux to.

If the *region* keyword is used, the *region-ID* selects which *region* to apply the source flux to.

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

No information of these fixes is written to *binary restart files*. None of the *fix_modify* options are relevant to these fixes. No global or per-atom quantities are stored by these fixes for access by various *output commands*. 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*.

2.46.5 Restrictions

These fixes are part of the DPD-MESO package. They are only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

Fix *edpd/source* must be used with the *pair_style edpd* command. Fix *tdpd/source* must be used with the *pair_style tdpd* command.

2.46.6 Related commands

pair_style edpd, *pair_style tdpd*, *compute edpd/temp/atom*, *compute tdpd/cc/atom*

2.46.7 Default

none

([Li2014](#)) Z. Li, Y.-H. Tang, H. Lei, B. Caswell and G.E. Karniadakis, “Energy-conserving dissipative particle dynamics with temperature-dependent properties”, J. Comput. Phys., 265: 113-127 (2014). DOI: 10.1016/j.jcp.2014.02.003

([Li2015](#)) Z. Li, A. Yazdani, A. Tartakovsky and G.E. Karniadakis, “Transport dissipative particle dynamics model for mesoscopic advection-diffusion-reaction problems”, J. Chem. Phys., 143: 014101 (2015). DOI: 10.1063/1.4923254

2.47 fix drag command

2.47.1 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)

2.47.2 Examples

```
fix center small-molecule drag 0.0 10.0 0.0 5.0 2.0
```

2.47.3 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.

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

No information about this fix is written to [binary restart files](#).

The [fix_modify respa](#) option is supported by this fix. This allows to set at which level of the [r-RESPA](#) integrator the fix is adding its forces. Default is the outermost level.

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](#).

2.47.5 Restrictions

This fix is part of the EXTRA-FIX package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

2.47.6 Related commands

fix spring, fix spring/self, fix spring/rg, fix smd

2.47.7 Default

none

2.48 fix drude command

2.48.1 Syntax

```
fix ID group-ID drude flag1 flag2 ... flagN
```

- ID, group-ID are documented in [fix](#) command
- drude = style name of this fix command
- flag1 flag2 ... flagN = Drude flag for each atom type (1 to N) in the system

2.48.2 Examples

```
fix 1 all drude 1 1 0 1 0 2 2 2  
fix 1 all drude C C N C N D D D
```

Example input scripts available: examples/PACKAGES/drude

2.48.3 Description

Assign each atom type in the system to be one of 3 kinds of atoms within the Drude polarization model. This fix is designed to be used with the [thermalized Drude oscillator model](#). Polarizable models in LAMMPS are described on the [Howto polarizable](#) doc page.

The three possible types can be designated with an integer (0,1,2) or capital letter (N,C,D):

- 0 or N = non-polarizable atom (not part of Drude model)
- 1 or C = Drude core
- 2 or D = Drude electron

2.48.4 Restrictions

This fix should be invoked before any other commands that implement the Drude oscillator model, such as [fix langevin/drude](#), [fix tgnvt/drude](#), [fix drude/transform](#), [compute temp/drude](#), [pair_style thole](#).

2.48.5 Related commands

`fix langevin/drude`, `fix tgnvt/drude`, `fix drude/transform`, `compute temp/drude`, `pair_style thole`

2.48.6 Default

none

2.49 fix drude/transform/direct command

2.50 fix drude/transform/inverse command

2.50.1 Syntax

```
fix ID group-ID style keyword value ...
```

- ID, group-ID are documented in `fix` command
- style = `drude/transform/direct` or `drude/transform/inverse`

2.50.2 Examples

```
fix 3 all drude/transform/direct
fix 1 all drude/transform/inverse
```

Example input scripts available: examples/PACKAGES/drude

2.50.3 Description

Transform the coordinates of Drude oscillators from real to reduced and back for thermalizing the Drude oscillators as described in ([Lamoureux](#)) using a Nose-Hoover thermostat. This fix is designed to be used with the [thermalized Drude oscillator model](#). Polarizable models in LAMMPS are described on the [Howto polarizable](#) doc page.

Drude oscillators are a pair of atoms representing a single polarizable atom. Ideally, the mass of Drude particles would vanish and their positions would be determined self-consistently by iterative minimization of the energy, the cores' positions being fixed. It is however more efficient and it yields comparable results, if the Drude oscillators (the motion of the Drude particle relative to the core) are thermalized at a low temperature. In that case, the Drude particles need a small mass.

The thermostats act on the reduced degrees of freedom, which are defined by the following equations. Note that in these equations upper case denotes atomic or center of mass values and lower case denotes Drude particle or dipole values. Primes denote the transformed (reduced) values, while bare letters denote the original values.

Masses:

$$M' = M + m$$

$$m' = \frac{M m}{M'}$$

Positions:

$$X' = \frac{MX + mx}{M'}$$

$$x' = x - X$$

Velocities:

$$V' = \frac{MV + mv}{M'}$$

$$v' = v - V$$

Forces:

$$F' = F + f$$

$$f' = \frac{Mf - mF}{M'}$$

This transform conserves the total kinetic energy

$$\frac{1}{2} (MV^2 + mv^2) = \frac{1}{2} (M'V'^2 + m'v'^2)$$

and the virial defined with absolute positions

$$X \cdot F + x \cdot f = X' \cdot F' + x' \cdot f'$$

This fix requires each atom know whether it is a Drude particle or not. You must therefore use the [fix drude](#) command to specify the Drude status of each atom type.

Note

only the Drude core atoms need to be in the group specified for this fix. A Drude electron will be transformed together with its core even if it is not itself in the group. It is safe to include Drude electrons or non-polarizable atoms in the group. The non-polarizable atoms will simply not be transformed.

This fix does NOT perform time integration. It only transform masses, coordinates, velocities and forces. Thus you must use separate time integration fixes, like [fix nve](#) or [fix npt](#) to actually update the velocities and positions of atoms. In order to thermalize the reduced degrees of freedom at different temperatures, two Nose-Hoover thermostats must be defined, acting on two distinct groups.

Note

The [fix drude/transform/direct](#) command must appear before any Nose-Hoover thermostatting fixes. The [fix drude/transform/inverse](#) command must appear after any Nose-Hoover thermostatting fixes.

Example:

```

fix fDIRECT all drude/transform/direct
fix fNVT gCORES nvt temp 300.0 300.0 100
fix fNVT gDRUDES nvt temp 1.0 1.0 100
fix finVERSE all drude/transform/inverse
compute TDRUDE all temp/drude
thermo_style custom step cpu etotal ke pe ebond ecoul elong press vol temp c_TDRUDE[1] c_
→ TDRUDE[2]

```

In this example, *gCORES* is the group of the atom cores and *gDRUDES* is the group of the Drude particles (electrons). The centers of mass of the Drude oscillators will be thermostatted at 300.0 and the internal degrees of freedom will be thermostatted at 1.0. The temperatures of cores and Drude particles, in center-of-mass and relative coordinates, are calculated using *compute temp/drude*

In addition, if you want to use a barostat to simulate a system at constant pressure, only one of the Nose-Hoover fixes must be *npt*, the other one should be *nvt*. You must add a *compute temp/com* and a *fix_modify* command so that the temperature of the *npt* fix be just that of its group (the Drude cores) but the pressure be the overall pressure *thermo_press*.

Example:

```

compute cTEMP_CORE gCORES temp/com
fix fDIRECT all drude/transform/direct
fix fNPT gCORES npt temp 298.0 298.0 100 iso 1.0 1.0 500
fix_modify fNPT temp cTEMP_CORE press thermo_press
fix fNVT gDRUDES nvt temp 5.0 5.0 100
fix finVERSE all drude/transform/inverse

```

In this example, *gCORES* is the group of the atom cores and *gDRUDES* is the group of the Drude particles. The centers of mass of the Drude oscillators will be thermostatted at 298.0 and the internal degrees of freedom will be thermostatted at 5.0. The whole system will be barostatted at 1.0.

In order to avoid the flying ice cube problem (irreversible transfer of linear momentum to the center of mass of the system), you may need to add a *fix momentum* command:

```
fix fMOMENTUM all momentum 100 linear 1 1 1
```

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

No information about this fix is written to *binary restart files*.

2.50.5 Restrictions

none

2.50.6 Related commands

fix drude, fix langevin/drude, compute temp/drude, pair_style thole

2.50.7 Default

none

(**Lamoureux**) Lamoureux and Roux, J Chem Phys, 119, 3025-3039 (2003).

2.51 fix dt/reset command

Accelerator Variants: *dt/reset/kk*

2.51.1 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 = re-compute 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 = *emax* or *units*

emax value = *Emax*

Emax = maximum kinetic energy change for an atom in one timestep (energy units)

units value = lattice or box

lattice = Xmax is defined in lattice units

box = Xmax is defined in simulation box units

2.51.2 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
fix 5 all dt/reset 5 NULL 0.001 0.5 emax 30 units box
```

2.51.3 Description

Reset the timestep size every N steps during a run, so that no atom moves further than the specified X_{max} distance, based on current atom velocities and forces. Optionally an additional criterion is imposed by the *emax* keyword, so that no atom's kinetic energy changes by more than the specified E_{max} .

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 X_{max} 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 X_{max} , though it may be smaller.

In addition if the *emax* keyword is used, the specified E_{max} value is enforced as a limit on how much an atom's kinetic energy can change. If the timestep required is even smaller than for the X_{max} displacement, then the smaller timestep is used.

Given this putative timestep for each atom, the minimum timestep value across all atoms is computed. Then the T_{min} and T_{max} 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.

Also note that the *dump_modify every/time* option allows dump files to be written at intervals specified by simulation time, rather than by timesteps. Simulation time is in time units; see the *units* doc page for details.

Styles with a *gpu*, *intel*, *kk*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed on the *Accelerator packages* page. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the GPU, INTEL, KOKKOS, OPENMP, and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the *Build package* page for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the *-suffix command-line switch* when you invoke LAMMPS, or you can use the *suffix* command in your input script.

See the *Accelerator packages* page for more instructions on how to use the accelerated styles effectively.

2.51.4 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*.

2.51.5 Restrictions

none

2.51.6 Related commands

timestep, *dump_modify every/time*

2.51.7 Default

The option defaults are units = lattice, and no emax kinetic energy limit.

2.52 fix efield command

2.53 fix efield/tip4p command

2.53.1 Syntax

```
fix ID group-ID style ex ey ez keyword value ...
```

- ID, group-ID are documented in *fix* command
- style = *efield* or *efield/tip4p*
- 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* or *potential*

region value = region-ID

region-ID = ID of region atoms must be in to have added force

energy value = v_name

v_name = variable with name that calculates the potential energy of each atom in the added
E-field

potential value = v_name

v_name = variable with name that calculates the electric potential of each atom in the added
E-field

2.53.2 Examples

```
fix kick external-field efield 1.0 0.0 0.0
fix kick external-field efield 0.0 0.0 v_oscillate
fix kick external-field efield/tip4p 1.0 0.0 0.0
```

2.53.3 Description

Add a force $\vec{F} = q\vec{E}$ 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.

Added in version 28Mar2023.

When the *efield/tip4p* style is used, the E-field will be applied to the position of the virtual charge site M of a TIP4P molecule instead of the oxygen position as it is defined by a corresponding *TIP4P pair style*. The forces on the M site due to the external field are projected on the oxygen and hydrogen atoms of the TIP4P molecules.

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 LAMMPS can compute the energy for each charged particle directly as

$$U_{efield} = -\vec{x} \cdot q\vec{E} = -q(x \cdot E_x + y \cdot E_y + z \cdot E_z),$$

so that $-\nabla U_{efield} = \vec{F}$. Similarly for point-dipole particles the energy can be computed as

$$U_{efield} = -\vec{\mu} \cdot \vec{E} = -\mu_x \cdot E_x + \mu_y \cdot E_y + \mu_z \cdot E_z$$

The *energy* keyword is optional if the added force is defined with one or more variables, and if you are performing dynamics via the *run* command. If the keyword is not used, LAMMPS will set the energy to 0.0, which is typically fine for dynamics.

The *energy* keyword (or *potential* keyword, described below) 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 LAMMPS 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 ensure that the formula defined for the atom-style *variable* is consistent with the force variable formulas, i.e. that $-\text{Grad}(E) = F$. 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.

Added in version 15Jun2023.

The *potential* keyword can be used as an alternative to the *energy* keyword to specify the name of an atom-style variable, which is used to compute the added electric potential to each atom as a function of its position. The variable should have units of electric field multiplied by distance (that is, in *units real*, the potential should be in volts). As with the *energy* keyword, the variable name is specified as “v_name”. The energy added by this fix is then calculated as the electric potential multiplied by charge.

The *potential* keyword is mainly intended for correct charge equilibration in simulations with *fix qeq/reaxff*, since with variable charges the electric potential can be known beforehand but the energy cannot. A small additional benefit is that the *energy* keyword requires an additional conversion to energy units which the *potential* keyword avoids. Thus, when the *potential* keyword is specified, the *energy* keyword must not be used. As with *energy*, the *potential* keyword is not allowed if the added field is a constant vector. The *potential* keyword is not supported by *fix efield/tip4p*.

2.53.4 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 global potential energy of the system as part of *thermodynamic output*. The default setting for this fix is *fix_modify energy no*. Note that this energy 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.

The *fix_modify virial* option is supported by this fix to add the contribution due to the added forces on atoms to both the global pressure and per-atom stress of the system via the *compute pressure* and *compute stress/atom* commands. The former can be accessed by *thermodynamic output*. The default setting for this fix is *fix_modify virial no*.

The *fix_modify respa* option is supported by this fix. This allows to set at which level of the *r-RESPA* integrator the fix adding its forces. Default is the outermost level.

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.

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.

Styles with a *gpu*, *intel*, *kk*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed on the *Accelerator packages*

page. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the GPU, INTEL, KOKKOS, OPENMP, and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the [Build package](#) page for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the [-suffix command-line switch](#) when you invoke LAMMPS, or you can use the [suffix](#) command in your input script.

See the [Accelerator packages](#) page for more instructions on how to use the accelerated styles effectively.

2.53.5 Restrictions

Fix style *efield/tip4p* is part of the EXTRA-FIX package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

Fix style *efield/tip4p* can only be used with tip4p pair styles.

2.53.6 Related commands

fix addforce

2.53.7 Default

none

2.54 fix ehex command

2.54.1 Syntax

```
fix ID group-ID ehex nevery F keyword value
```

- ID, group-ID are documented in [fix](#) command
- ehex = style name of this fix command
- nevery = add/subtract heat every this many timesteps
- F = energy flux into the reservoir (energy/time units)
- zero or more keyword/value pairs may be appended to args
- keyword = *region* or *constraint* or *com* or *hex*

region value = region-ID

region-ID = ID of region (reservoir) atoms must be in for added thermostating force
constraint value = none

apply the constraint algorithm (SHAKE or RATTLE) again at the end of the timestep
com value = none

rescale all sites of a constrained cluster of atom if its COM is in the reservoir

hex value = none

omit the coordinate correction to recover the HEX algorithm

2.54.2 Examples

```
# Lennard-Jones, from examples/in.ehex.lj

fix fnve all nve
# specify regions rhot and rcold
...
fix fhot all ehex 1 0.15 region rhot
fix fcold all ehex 1 -0.15 region rcold

# SPC/E water, from examples/in.ehex.spce
fix fnve all nve
# specify regions rhot and rcold
...
fix fhot all ehex 1 0.075 region rhot constrain com
fix fcold all ehex 1 -0.075 region rcold constrain com
fix frattle all rattle 1e-10 400 0 b 1 a 1
```

2.54.3 Description

This fix implements the asymmetric version of the enhanced heat exchange algorithm ([Wirnsberger](#)). The eHEX algorithm is an extension of the heat exchange algorithm ([Ikeshoji](#)) and adds an additional coordinate integration to account for higher-order truncation terms in the operator splitting. The original HEX algorithm (implemented as [fix heat](#)) is known to exhibit a slight energy drift limiting the accessible simulation times to a few nanoseconds. This issue is greatly improved by the new algorithm decreasing the energy drift by at least a factor of a hundred (LJ and SPC/E water) with little computational overhead.

In both algorithms (non-translational) kinetic energy is constantly swapped between regions (reservoirs) to impose a heat flux onto the system. The equations of motion are therefore modified if a particle i is located inside a reservoir Γ_k where $k > 0$. We use Γ_0 to label those parts of the simulation box which are not thermostatted.) The input parameter *region-ID* of this fix corresponds to k . The energy swap is modelled by introducing an additional thermostating force to the equations of motion, such that the time evolution of coordinates and momenta of particle i becomes ([Wirnsberger](#))

$$\begin{aligned}\dot{\mathbf{r}}_i &= \mathbf{v}_i, \\ \dot{\mathbf{v}}_i &= \frac{\mathbf{f}_i}{m_i} + \frac{\mathbf{g}_i}{m_i}.\end{aligned}$$

The thermostating force is given by

$$\mathbf{g}_i = \begin{cases} \frac{m_i}{2} \frac{F_{\Gamma_k(\mathbf{r}_i)}}{K_{\Gamma_k(\mathbf{r}_i)}} (\mathbf{v}_i - \mathbf{v}_{\Gamma_k(\mathbf{r}_i)}) & k(\mathbf{r}_i) > 0 \text{ (inside a reservoir),} \\ 0 & \text{otherwise,} \end{cases}$$

where m_i is the mass and $k(\mathbf{r}_i)$ maps the particle position to the respective reservoir. The quantity $F_{\Gamma_k(\mathbf{r}_i)}$ corresponds to the input parameter F , which is the energy flux into the reservoir. Furthermore, $K_{\Gamma_k(\mathbf{r}_i)}$ and $\mathbf{v}_{\Gamma_k(\mathbf{r}_i)}$ denote the non-translational kinetic energy and the center of mass velocity of that reservoir. The thermostating force does not affect the center of mass velocities of the individual reservoirs and the entire simulation box. A derivation of the equations and details on the numerical implementation with velocity Verlet in LAMMPS can be found in reference “([Wirnsberger](#))”#_Wirnsberger.

Note

This fix only integrates the thermostating force and must be combined with another integrator, such as [fix nve](#), to solve the full equations of motion.

This fix is different from a thermostat such as `fix nvt` or `fix temp/rescale` in that energy is added/subtracted continually. Thus if there is not another mechanism in place to counterbalance this effect, the entire system will heat or cool continuously.

Note

If heat is subtracted from the system too aggressively so that the group's kinetic energy would go to zero, then LAMMPS will halt with an error message. Increasing the value of `nevery` means that heat is added/subtracted less frequently but in larger portions. The resulting temperature profile will therefore be the same.

This fix will default to `fix_hex` (HEX algorithm) if the keyword `hex` is specified.

Compatibility with SHAKE and RATTLE (rigid molecules):

This fix is compatible with `fix shake` and `fix rattle`. If either of these constraining algorithms is specified in the input script and the keyword `constrain` is set, the bond distances will be corrected a second time at the end of the integration step. It is recommended to specify the keyword `com` in addition to the keyword `constrain`. With this option all sites of a constrained cluster are rescaled, if its center of mass is located inside the region. Rescaling all sites of a cluster by the same factor does not introduce any velocity components along fixed bonds. No rescaling takes place if the center of mass lies outside the region.

Note

You can only use the keyword `com` along with `constrain`.

To achieve the highest accuracy it is recommended to use `fix rattle` with the keywords `constrain` and `com` as shown in the second example. Only if RATTLE is employed, the velocity constraints will be satisfied.

Note

Even if RATTLE is used and the keywords `com` and `constrain` are both set, the coordinate constraints will not necessarily be satisfied up to the target precision. The velocity constraints are satisfied as long as all sites of a cluster are rescaled (keyword `com`) and the cluster does not span adjacent reservoirs. The current implementation of the eHEX algorithm introduces a small error in the bond distances, which goes to zero with order three in the timestep. For example, in a simulation of SPC/E water with a timestep of 2 fs the maximum relative error in the bond distances was found to be on the order of 10^{-7} for relatively large temperature gradients. A higher precision can be achieved by decreasing the timestep.

2.54.4 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 parameter of this fix can be used with the `start/stop` keywords of the `run` command. This fix is not invoked during `energy minimization`.

2.54.5 Restrictions

This fix is part of the RIGID package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

2.54.6 Related commands

fix heat, fix thermal/conductivity, compute temp, compute temp/region

2.54.7 Default

none

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

(Wirnsberger) Wirnsberger, Frenkel, and Dellago, J Chem Phys, 143, 124104 (2015).

2.55 fix electrode/conp command

Accelerator Variant: *electrode/conp/intel*

2.56 fix electrode/conq command

Accelerator Variant: *electrode/conq/intel*

2.57 fix electrode/thermo command

Accelerator Variant: *electrode/thermo/intel*

2.57.1 Syntax

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

- ID, group-ID are documented in [fix](#) command
- style = *electrode/conp* or *electrode/conq* or *electrode/thermo*
- args = arguments used by a particular style
 - electrode/conp* args = potential eta
 - electrode/conq* args = charge eta
 - electrode/thermo* args = potential eta temp values
 - potential = electrode potential
 - charge = electrode charge
 - eta = reciprocal width of electrode charge smearing (can be NULL if eta keyword is used)
 - temp values = T_v tau_v rng_v
 - T_v = temperature of thermo-potentiostat

`tau_v` = time constant of thermo-potentiostat
`rng_v` = integer used to initialize random number generator

- zero or more keyword/value pairs may be appended
- keyword = *algo* or *symm* or *couple* or *etypes* or *ffield* or *write_mat* or *write_inv* or *read_mat* or *read_inv* or *qtotal* or *eta*

`algo` values = `mat_inv` or `mat_cg tol` or `cg tol`
specify the algorithm used to compute the electrode charges
`symm` value = `on` or `off`
turn on/off charge neutrality constraint for the electrodes
`couple` values = group-ID val
group-ID = group of atoms treated as additional electrode
`val` = electric potential or charge on this electrode
`etypes` value = `on` or `off`
turn on/off type-based optimized neighbor lists (electrode and electrolyte types may not overlap)
`ffield` value = `on` or `off`
turn on/off finite-field implementation
`write_mat` value = filename
filename = file to which to write elastance matrix
`write_inv` value = filename
filename = file to which to write inverted matrix
`read_mat` value = filename
filename = file from which to read elastance matrix
`read_inv` value = filename
filename = file from which to read inverted matrix
`qtotal` value = number or `v_equal`-style variable
add overall potential so that all electrode charges add up to `qtotal`
`eta` value = d_propname
`d_propname` = a custom double vector defined via fix property/atom

2.57.2 Examples

```
fix fxcomp bot electrode/comp -1.0 1.805 couple top 1.0 couple ref 0.0 write_inv inv.csv symm on
fix fxcomp electrodes electrode/conq 0.0 1.805 algo cg 1e-5
fix fxcomp bot electrode/thermo -1.0 1.805 temp 298 100 couple top 1.0
```

2.57.3 Description

The *electrode* fixes implement the constant potential method (CPM) (*Siepmann, Reed*), and modern variants, to accurately model electrified, conductive electrodes. This is primarily useful for studying electrode-electrolyte interfaces, especially at high potential differences or ionicities, with non-planar electrodes such as nanostructures or nanopores, and to study dynamic phenomena such as charging or discharging time scales or conductivity or ionic diffusivities.

Each *electrode* fix allows users to set additional electrostatic relationships between the specified groups which model useful electrostatic configurations:

- *electrode/comp* sets potentials or potential differences between electrodes
 - (resulting in changing electrode total charges)
- *electrode/conq* sets the total charge on each electrode
 - (resulting in changing electrode potentials)

- *electrode/thermo* sets a thermopotentiostat ([Deissenbeck](#)) between two electrodes
 - (resulting in changing charges and potentials with appropriate average potential difference and thermal variance)

The first group-ID provided to each fix specifies the first electrode group, and more group(s) are added using the *couple* keyword for each additional group. While *electrode/thermo* only accepts two groups, *electrode/conp* and *electrode/conq* accept any number of groups, up to LAMMPS's internal restrictions (see Restrictions below). Electrode groups must not overlap, i.e. the fix will issue an error if any particle is detected to belong to at least two electrode groups.

CPM involves updating charges on groups of electrode particles, per time step, so that the system's total energy is minimized with respect to those charges. From basic electrostatics, this is equivalent to making each group conductive, or imposing an equal electrostatic potential on every particle in the same group (hence the name CPM). The charges are usually modelled as a Gaussian distribution to make the charge-charge interaction matrix invertible ([Gingrich](#)). The keyword *eta* specifies the distribution's width in units of inverse length.

Added in version 22Dec2022.

Three algorithms are available to minimize the energy, varying in how matrices are pre-calculated before a run to provide computational speedup. These algorithms can be selected using the keyword *algo*:

- *algo mat_inv* pre-calculates the capacitance matrix and obtains the charge configuration in one matrix-vector calculation per time step
- *algo mat_cg* pre-calculates the elastance matrix (inverse of capacitance matrix) and obtains the charge configuration using a conjugate gradient solver in multiple matrix-vector calculations per time step
- *algo cg* does not perform any pre-calculation and obtains the charge configuration using a conjugate gradient solver and multiple calculations of the electric potential per time step.

For both *cg* methods, the command must specify the conjugate gradient tolerance. *fix electrode/thermo* currently only supports the *mat_inv* algorithm.

The keyword *symm* can be set *on* (or *off*) to turn on (or turn off) the capacitance matrix constraint that sets total electrode charge to be zero. This has slightly different effects for each fix *electrode* variant. For *fix electrode/conp*, with *symm off*, the potentials specified are absolute potentials, but the charge configurations satisfying them may add up to an overall non-zero, varying charge for the electrodes (and thus the simulation box). With *symm on*, the total charge over all electrode groups is constrained to zero, and potential differences rather than absolute potentials are the physically relevant quantities.

For *fix electrode/conq*, with *symm off*, overall neutrality is explicitly obeyed or violated by the user input (which is not checked!). With *symm on*, overall neutrality is ensured by ignoring the user-input charge for the last listed electrode (instead, its charge will always be minus the total sum of all other electrode charges). For *fix electrode/thermo*, overall neutrality is always automatically imposed for any setting of *symm*, but *symm on* allows finite-field mode (*ffield on*, described below) for faster simulations.

For all three fixes, any potential (or charge for *conq*) can be specified as an equal-style variable prefixed with “v_”. For example, the following code will ramp the potential difference between electrodes from 0.0V to 2.0V over the course of the simulation:

```
fix fxconp bot electrode/conp 0.0 1.805 couple top v_v symm on
variable v equal ramp(0.0, 2.0)
```

Note that these fixes only parse their supplied variable name when starting a run, and so these fixes will accept equal-style variables defined *after* the fix definition, including variables dependent on the fix's own output. This is useful, for example, in the fix's internal finite-field commands (see below). For an advanced example of this see the *in.conq2* input file in the directory *examples/PACKAGES/electrode/graph-il*.

This fix necessitates the use of a long range solver that calculates and provides the matrix of electrode-electrode interactions and a vector of electrode-electrolyte interactions. The Kspace styles *ewald/electrode*, *pppm/electrode* and *pppm/electrode/intel* are created specifically for this task ([Ahrens-Iwers](#)).

For systems with non-periodic boundaries in one or two directions dipole corrections are available with the `kspace_modify`. For ewald/electrode a two-dimensional Ewald summation (`Hu`) can be used by setting “slab ew2d”:

```
kspace_modify slab <slab_factor>
kspace_modify wire <wire_factor>
kspace_modify slab ew2d
```

Two implementations for the calculation of the elastance matrix are available with pppm and can be selected using the `amat onestep/twostep` keyword. `onestep` is the default; `twostep` can be faster for large electrodes and a moderate mesh size but requires more memory.

```
kspace_modify amat onestep/twostep
```

For all versions of the fix, the keyword-value `ffield on` enables the finite-field mode ([Dufils](#), [Tee](#)), which uses an electric field across a periodic cell instead of non-periodic boundary conditions to impose a potential difference between the two electrodes bounding the cell. The fix (with name `fix-ID`) detects which of the two electrodes is “on top” (has the larger maximum z -coordinate among all particles). Assuming the first electrode group is on top, it then issues the following commands internally:

```
variable fix-ID_field_zfield equal (f_fix-ID[2]-f_fix-ID[1])/lz
efield fix-ID_efield all efield 0.0 0.0 v_fix-ID_ffield_zfield
```

which implements the required electric field as the potential difference divided by cell length. The internal commands use variable so that the electric field will correctly vary with changing potentials in the correct way (for example with equal-style potential difference or with `fix electrode/conq`). This keyword requires two electrodes and will issue an error with any other number of electrodes. This keyword requires electroneutrality to be imposed (`symm on`) and will issue an error otherwise.

Changed in version 22Dec2022.

For all versions of the fix, the keyword-value `etypes on` enables type-based optimized neighbor lists. With this feature enabled, LAMMPS provides the fix with an occasional neighbor list restricted to electrode-electrode interactions for calculating the electrode matrix, and a perpetual neighbor list restricted to electrode-electrolyte interactions for calculating the electrode potentials, using particle types to list only desired interactions, and typically resulting in 5–10% less computational time. Without this feature the fix will simply use the active pair style’s neighbor list. This feature cannot be enabled if any electrode particle has the same type as any electrolyte particle (which would be unusual in a typical simulation) and the fix will issue an error in that case.

Added in version 17Apr2024.

The keyword `qtotal` causes `fix electrode/conp` and `fix electrode/thermo` to add an overall potential to all electrodes so that the total charge on the electrodes is a specified amount (which may be an equal-style variable). For example, if a user wanted to simulate a solution of excess cations such that the total electrolyte charge is +2, setting `qtotal -2` would cause the total electrode charge to be -2, so that the simulation box remains overall electroneutral. Since `fix electrode/conq` constrains the total charges of individual electrodes, and since `symm on` constrains the total charge of all electrodes to be zero, either option is incompatible with the `qtotal` keyword (even if `qtotal` is set to zero).

Added in version 17Apr2024.

The keyword `eta` takes the name of a custom double vector defined via `fix property/atom`. The values will be used instead of the standard eta value. The `property/atom` fix must be for vector of double values and use the `ghost on` option.

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

This fix currently does not write any information to restart files.

The *fix_modify tf* option enables the Thomas-Fermi metallicity model (*Scalfi*) and allows parameters to be set for each atom type.

`fix_modify ID tf type length voronoi`

If this option is used, these two parameters must be set for all atom types of the electrode:

- *tf* is the Thomas-Fermi length l_{TF}
- *voronoi* is the Voronoi volume per atom in units of length cubed

Different types may have different *tf* and *voronoi* values. The following self-energy term is then added for all electrode atoms:

$$A_{ii} = \frac{1}{4\pi\varepsilon_0} \times \frac{4\pi l_{TF}^2}{\text{Voronoi volume}}$$

The *fix_modify timer* option turns on (off) additional timer outputs in the log file, for code developers to track optimization.

`fix_modify ID timer on/off`

These fixes compute a global (extensive) scalar, a global (intensive) vector, and a global array, which can be accessed by various *output commands*.

The global scalar outputs the energy added to the system by this fix, which is the negative of the total charge on each electrode multiplied by that electrode's potential.

The global vector outputs the potential on each electrode (and thus has N entries if the fix manages N electrode groups), in *units* of electric field multiplied by distance (thus volts for *real* and *metal* units). The electrode groups' ordering follows the order in which they were input in the fix command using *couple*. The global vector output is useful for *fix electrode/conq* and *fix electrode/thermo*, where potential is dynamically updated based on electrolyte configuration instead of being directly set.

The global array has N rows and $2N+1$ columns, where the fix manages N electrode groups managed by the fix. For the I -th row of the array, the elements are:

- $\text{array}[I][1]$ = total charge that group I would have had *if it were at 0 V applied potential*
- $\text{array}[I][2 \text{ to } N+1]$ = the N entries of the I -th row of the electrode capacitance matrix (definition follows)
- $\text{array}[I][N+2 \text{ to } 2N+1]$ = the N entries of the I -th row of the electrode elastance matrix (the inverse of the electrode capacitance matrix)

The $N \times N$ electrode capacitance matrix, denoted \mathbf{C} in the following equation, summarizes how the total charge induced on each electrode (\mathbf{Q} as an N -vector) is related to the potential on each electrode, \mathbf{V} , and the charge-at-0V \mathbf{Q}_{0V} (which is influenced by the local electrolyte structure):

$$\mathbf{Q} = \mathbf{Q}_{0V} + \mathbf{C} \cdot \mathbf{V}$$

The charge-at-0V, electrode capacitance and elastance matrices are internally used to calculate the potentials required to induce the specified total electrode charges in *fix electrode/conq* and *fix electrode/thermo*. With the *symm on* option, the electrode capacitance matrix would be singular, and thus its last row is replaced with N copies of its top-left entry (\mathbf{C}_{11}) for invertibility.

The global array output is mainly useful for quickly determining the ‘vacuum capacitance’ of the system (capacitance with only electrodes, no electrolyte), and can also be used for advanced simulations setting the potential as some function of the charge-at-0V (such as the `in.cong2` example mentioned above).

Please cite ([Ahrens-Iwers2022](#)) in any publication that uses this implementation. Please cite also the publication on the combination of the CPM with PPPM if you use `pppm/electrode` ([Ahrens-Iwers](#)).

2.57.5 Restrictions

For algorithms that use a matrix for the electrode-electrode interactions, positions of electrode particles have to be immobilized at all times.

With *field off* (i.e. the default), the box geometry is expected to be *z*-non-periodic (i.e. *boundary p pf*), and this fix will issue an error if the box is *z*-periodic. With *field on*, the box geometry is expected to be *z*-periodic, and this fix will issue an error if the box is *z*-non-periodic.

The parallelization for the fix works best if electrode atoms are evenly distributed across processors. For a system with two electrodes at the bottom and top of the cell this can be achieved with *processors ** 2*, or with the line

```
if "${extract_setting(world_size) % 2} == 0" then "processors ** 2"
```

which avoids an error if the script is run on an odd number of processors (such as on just one processor for testing).

The fix creates an additional group named *[fix-ID]_group* which is the union of all electrode groups supplied to LAMMPS. This additional group counts towards LAMMPS’s limitation on the total number of groups (currently 32), which may not allow scripts that use that many groups to run with this fix.

The matrix-based algorithms (*algo mat_inv* and *algo mat_cg*) currently store an interaction matrix (either elastance or capacitance) of N by N doubles for each MPI process. This memory requirement may be prohibitive for large electrode groups. The fix will issue a warning if it expects to use more than 0.5 GiB of memory.

2.57.6 Default

The default keyword-option settings are *algo mat_inv*, *symm off*, *etypes off* and *ffield off*.

Styles with a *gpu*, *intel*, *kk*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed on the [Accelerator packages](#) page. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the GPU, INTEL, KOKKOS, OPENMP, and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the [Build package](#) page for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the *-suffix command-line switch* when you invoke LAMMPS, or you can use the *suffix* command in your input script.

See the [Accelerator packages](#) page for more instructions on how to use the accelerated styles effectively.

(Siepmann) Siepmann and Sprik, J. Chem. Phys. 102, 511 (1995).

(Reed) Reed *et al.*, J. Chem. Phys. 126, 084704 (2007).

(Deissenbeck) Deissenbeck *et al.*, Phys. Rev. Letters 126, 136803 (2021).

(Gingrich) Gingrich, *MSc thesis* <<https://gingrich.chem.northwestern.edu/papers/ThesiswCorrections.pdf>>` (2010).

(Ahrens-Iwers) Ahrens-Iwers and Meissner, *J. Chem. Phys.* 155, 104104 (2021).

(Hu) Hu, *J. Chem. Theory Comput.* 10, 5254 (2014).

(Dufils) Dufils *et al.*, *Phys. Rev. Letters* 123, 195501 (2019).

(Tee) Tee and Searles, *J. Chem. Phys.* 156, 184101 (2022).

(Scalfi) Scalfi *et al.*, *J. Chem. Phys.*, 153, 174704 (2020).

(Ahrens-Iwers2022) Ahrens-Iwers *et al.*, *J. Chem. Phys.* 157, 084801 (2022).

2.58 fix electron/stopping command

2.59 fix electron/stopping/fit command

2.59.1 Syntax

```
fix ID group-ID style args
```

- ID, group-ID are documented in [fix](#) command
- style = *electron/stopping* or *electron/stopping/fit*

electron/stopping args = Ecut file keyword value ...

Ecut = minimum kinetic energy for electronic stopping (energy units)

file = name of the file containing the electronic stopping power table

electron/stopping/fit args = Ecut c1 c2 ...

Ecut = minimum kinetic energy for electronic stopping (energy units)

c1 c2 = linear and quadratic coefficients for the fitted quadratic polynomial

- zero or more keyword/value pairs may be appended to args for style = *electron/stopping*

keyword = region or minneigh

region value = region-ID

region-ID = region whose atoms will be affected by this fix

minneigh value = minneigh

minneigh = minimum number of neighbors an atom to have stopping applied

2.59.2 Examples

```
fix el all electron/stopping 10.0 elstop-table.txt
fix el all electron/stopping 10.0 elstop-table.txt minneigh 3
fix el mygroup electron/stopping 1.0 elstop-table.txt region bulk
fix 1 all electron/stopping/fit 4.63 3.3e-3 4.0e-8
fix 1 all electron/stopping/fit 3.49 1.8e-3 9.0e-8 7.57 4.2e-3 5.0e-8
```

2.59.3 Description

This fix implements inelastic energy loss for fast projectiles in solids. It applies a friction force to fast moving atoms to slow them down due to *electronic stopping* (energy lost via electronic collisions per unit of distance). This fix should be used for simulation of irradiation damage or ion implantation, where the ions can lose noticeable amounts of energy from electron excitations. If the electronic stopping power is not considered, the simulated range of the ions can be severely overestimated (*Nordlund98*, *Nordlund95*).

The electronic stopping is implemented by applying a friction force to each atom as:

$$\vec{F}_i = \vec{F}_i^0 - \frac{\vec{v}_i}{\|\vec{v}_i\|} \cdot S_e$$

where \vec{F}_i is the resulting total force on the atom. \vec{F}_i^0 is the original force applied to the atom, \vec{v}_i is its velocity and S_e is the stopping power of the ion.

Note

In addition to electronic stopping, atomic cascades and irradiation simulations require the use of an adaptive timestep (see *fix dt/reset*) and the repulsive ZBL potential (see *ZBL* potential) or similar. Without these settings the interaction between the ion and the target atoms will be faulty. It is also common to use in such simulations a thermostat (*fix_nvt*) in the borders of the simulation cell.

Note

This fix removes energy from fast projectiles without depositing it as a heat to the simulation cell. Such implementation might lead to the unphysical results when the amount of energy deposited to the electronic system is large, e.g. simulations of Swift Heavy Ions (energy per nucleon of 100 keV/amu or higher) or multiple projectiles. You could compensate energy loss by coupling bulk atoms with some thermostat or control heat transfer between electronic and atomic subsystems with the two-temperature model (*fix_ttm*).

At low velocities the electronic stopping is negligible. The electronic friction is not applied to atoms whose kinetic energy is smaller than *Ecut*, or smaller than the lowest energy value given in the table in *file*. Electronic stopping should be applied only when a projectile reaches bulk material. This fix scans neighbor list and excludes atoms with fewer than *minneigh* neighbors (by default one). If the pair potential cutoff is large, *minneigh* should be increased, though not above the number of nearest neighbors in bulk material. An alternative is to disable the check for neighbors by setting *minneigh* to zero and using the *region* keyword. This is necessary when running simulations of cluster bombardment.

If the *region* keyword is used, the atom must also be in the specified geometric *region* in order to have electronic stopping applied to it. This is useful if the position of the bulk material is fixed. By default the electronic stopping is applied everywhere in the simulation cell.

The energy ranges and stopping powers are read from the file *file*. Lines starting with # and empty lines are ignored. Otherwise each line must contain exactly **N+1** numbers, where **N** is the number of atom types in the simulation.

The first column is the energy for which the stopping powers on that line apply. The energies must be sorted from the smallest to the largest. The other columns are the stopping powers S_e for each atom type, in ascending order, in force *units*. The stopping powers for intermediate energy values are calculated with linear interpolation between 2 nearest points.

For example:

```
# This is a comment
#      atom-1   atom-2
# eV    eV/Ang   eV/Ang # units metal
 10      0       0
250     60      80
750    100     150
```

If an atom which would have electronic stopping applied to it has a kinetic energy higher than the largest energy given in file, LAMMPS will exit with an error message.

The stopping power depends on the energy of the ion and the target material. The electronic stopping table can be obtained from scientific publications, experimental databases or by using [SRIM](#) software. Other programs such as [CasP](#) or [PASS](#) can calculate the energy deposited as a function of the impact parameter of the ion; these results can be used to derive the stopping power.

Style *electron/stopping/fit* calculates the electronic stopping power and cumulative energy lost to the electron gas via a quadratic functional and applies a drag force to the classical equations-of-motion for all atoms moving above some minimum cutoff velocity (i.e., kinetic energy). These coefficients can be determined by fitting a quadratic polynomial to electronic stopping data predicted by, for example, SRIM or TD-DFT. Multiple ‘Ecut c1 c2’ values can be provided for multi-species simulations in the order of the atom types. There is an examples/PACKAGES/electron_stopping/ directory, which illustrates uses of this command. Details of this implementation are further described in [Stewart2018](#) and [Lee2020](#).

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

No information about this fix is written to *binary restart files*.

The *fix_modify* options are not supported.

This fix computes a global scalar, which can be accessed by various *output commands*. The scalar is the total energy loss from electronic stopping applied by this fix since the start of the latest run. It is considered “intensive”.

The *start/stop* keywords of the *run* command have no effect on this fix.

2.59.5 Restrictions

This pair style is part of the EXTRA-FIX package. It is only enabled if LAMMPS was built with that package. See the *Build package* doc page for more info.

2.59.6 Default

The default is no limitation by region, and minneigh = 1.

(**electronic stopping**) Wikipedia - Electronic Stopping Power: https://en.wikipedia.org/wiki/Stopping_power_%28particle_radiation%29

(**Nordlund98**) Nordlund, Kai, et al. Physical Review B 57.13 (1998): 7556.

(**Nordlund95**) Nordlund, Kai. Computational materials science 3.4 (1995): 448-456.

(**SRIM**) SRIM webpage: <http://www.srim.org/>

(**CasP**) CasP webpage: <http://www.casp-program.org/>

(PASS) PASS webpage: <https://www.sdu.dk/en/DPASS>

(Stewart2018) J.A. Stewart, et al. (2018) Journal of Applied Physics, 123(16), 165902.

(Lee2020) C.W. Lee, et al. (2020) Physical Review B, 102(2), 024107.

2.60 fix enforce2d command

Accelerator Variants: *enforce2d/kk*

2.60.1 Syntax

```
fix ID group-ID enforce2d
```

- ID, group-ID are documented in [fix](#) command
- enforce2d = style name of this fix command

2.60.2 Examples

```
fix 5 all enforce2d
```

2.60.3 Description

Zero out the z-dimension velocity and force on each atom in the group. This is useful when running a 2d simulation to ensure that atoms do not move from their initial z coordinate.

Styles with a *gpu*, *intel*, *kk*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed on the [Accelerator packages](#) page. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the GPU, INTEL, KOKKOS, OPENMP, and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the [Build package](#) page for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the [-suffix command-line switch](#) when you invoke LAMMPS, or you can use the [suffix](#) command in your input script.

See the [Accelerator packages](#) page for more instructions on how to use the accelerated styles effectively.

2.60.4 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.

2.60.5 Restrictions

none

2.60.6 Related commands

none

2.60.7 Default

none

2.61 fix eos/cv command

2.61.1 Syntax

```
fix ID group-ID eos/cv cv
```

- ID, group-ID are documented in [fix](#) command
- eos/cv = style name of this fix command
- cv = constant-volume heat capacity (energy/temperature units)

2.61.2 Examples

```
fix 1 all eos/cv 0.01
```

2.61.3 Description

Fix *eos/cv* applies a mesoparticle equation of state to relate the particle internal energy (u_i) to the particle internal temperature (θ_i). The *eos/cv* mesoparticle equation of state requires the constant-volume heat capacity, and is defined as follows:

$$u_i = u_i^{mech} + u_i^{cond} = C_V \theta_i$$

where C_V is the constant-volume heat capacity, u^{cond} is the internal conductive energy, and u^{mech} is the internal mechanical energy. Note that alternative definitions of the mesoparticle equation of state are possible.

2.61.4 Restrictions

This command is part of the DPD-REACT package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

This command also requires use of the [atom_style dpd](#) command.

2.61.5 Related commands

[fix shardlow](#), [pair dpd/fdt](#)

2.61.6 Default

none

(Larentzos) J.P. Larentzos, J.K. Brennan, J.D. Moore, and W.D. Mattson, “LAMMPS Implementation of Constant Energy Dissipative Particle Dynamics (DPD-E)”, ARL-TR-6863, U.S. Army Research Laboratory, Aberdeen Proving Ground, MD (2014).

2.62 fix eos/table command

2.62.1 Syntax

```
fix ID group-ID eos/table style file N keyword
```

- ID, group-ID are documented in [fix](#) command
- eos/table = style name of this fix command
- style = *linear* = method of interpolation
- file = filename containing the tabulated equation of state
- N = use N values in *linear* tables
- keyword = name of table keyword corresponding to table file

2.62.2 Examples

```
fix 1 all eos/table linear eos.table 100000 KEYWORD
```

2.62.3 Description

Fix *eos/table* applies a tabulated mesoparticle equation of state to relate the particle internal energy (*u_i*) to the particle internal temperature (*dpdTheta_i*).

Fix *eos/table* creates interpolation tables of length *N* from internal energy values listed in a file as a function of internal temperature.

The interpolation tables are created by fitting cubic splines to the file values and interpolating energy values at each of *N* internal temperatures, and vice versa. During a simulation, these tables are used to interpolate internal energy or temperature values as needed. The interpolation is done with the *linear* style.

For the *linear* style, the internal temperature is used to find 2 surrounding table values from which an internal energy is computed by linear interpolation, and vice versa.

The filename specifies a file containing tabulated internal temperature and internal energy values. The keyword specifies a section of the file. The format of this file is described below.

The format of a tabulated file is as follows (without the parenthesized comments):

```
# EOS TABLE      (one or more comment or blank lines)

KEYWORD      (keyword is first text on line)
N 500        (N parameter)
             (blank)
1  1.00 0.000 (index, internal temperature, internal energy)
2  1.02 0.001
...
500 10.0 0.500
```

A section begins with a non-blank line whose first character is not a “#”; blank lines or lines starting with “#” can be used as comments between sections. The first line begins with a keyword which identifies the section. The line can contain additional text, but the initial text must match the argument specified in the fix command.

The next line lists the number of table entries. The parameter “N” is required and its value is the number of table entries that follow. Note that this may be different than the *N* specified in the [fix eos/table](#) command. Let *Ntable* = *N* in the fix command, and *Nfile* = “*N*” in the tabulated file. What LAMMPS does is a preliminary interpolation by creating splines using the *Nfile* tabulated values as nodal points. It uses these to interpolate as needed to generate energy and temperature values at *Ntable* different points. The resulting tables of length *Ntable* are then used as described above, when computing energy and temperature relationships. This means that if you want the interpolation tables of length *Ntable* to match exactly what is in the tabulated file (with effectively no preliminary interpolation), you should set *Ntable* = *Nfile*.

Following a blank line, the next *N* lines list the tabulated values. On each line, the first value is the index from 1 to *N*, the second value is the internal temperature (in temperature units), the third value is the internal energy (in energy units).

Note that the internal temperature and internal energy values must increase from one line to the next.

Note that one file can contain many sections, each with a tabulated potential. LAMMPS reads the file section by section until it finds one that matches the specified keyword.

2.62.4 Restrictions

This command is part of the DPD-REACT package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

This command also requires use of the [atom_style dpd](#) command.

The equation of state must be a monotonically increasing function.

An error will occur if the internal temperature or internal energies are not within the table cutoffs.

2.62.5 Related commands

[fix shardlow](#), [pair dpd/fdt](#)

2.62.6 Default

none

2.63 fix eos/table/rx command

Accelerator Variants: *eos/table/rx/kk*

2.63.1 Syntax

```
fix ID group-ID eos/table/rx style file1 N keyword ...
```

- ID, group-ID are documented in [fix](#) command
- eos/table/rx = style name of this fix command
- style = *linear* = method of interpolation
- file1 = filename containing the tabulated equation of state
- N = use N values in *linear* tables
- keyword = name of table keyword corresponding to table file
- file2 = filename containing the heats of formation of each species (optional)
- deltaHf = heat of formation for a single species in energy units (optional)
- energyCorr = energy correction in energy units (optional)
- tempCorrCoeff = temperature correction coefficient (optional)

2.63.2 Examples

```
fix 1 all eos/table/rx linear eos.table 10000 KEYWORD thermo.table
fix 1 all eos/table/rx linear eos.table 10000 KEYWORD 1.5
fix 1 all eos/table/rx linear eos.table 10000 KEYWORD 1.5 0.025 0.0
```

2.63.3 Description

Fix *eos/table/rx* applies a tabulated mesoparticle equation of state to relate the concentration-dependent particle internal energy (u_i) to the particle internal temperature (θ_i).

The concentration-dependent particle internal energy (u_i) is computed according to the following relation:

$$U_i = \sum_{j=1}^m c_{i,j}(u_j + \Delta H_{f,j}) + \frac{3k_B T}{2} + Nk_B T$$

where m is the number of species, $c_{i,j}$ is the concentration of species j in particle i , u_j is the internal energy of species j , $\Delta H_{f,j}$ is the heat of formation of species j , N is the number of molecules represented by the coarse-grained particle, k_B is the Boltzmann constant, and T is the temperature of the system. Additionally, it is possible to modify the concentration-dependent particle internal energy relation by adding an energy correction, temperature-dependent correction, and/or a molecule-dependent correction. An energy correction can be specified as a constant (in energy units). A temperature correction can be specified by multiplying a temperature correction coefficient by the internal temperature. A molecular correction can be specified by multiplying a molecule correction coefficient by the average number of product gas particles in the coarse-grain particle.

Fix *eos/table/rx* creates interpolation tables of length N from m internal energy values of each species u_j listed in a file as a function of internal temperature. During a simulation, these tables are used to interpolate internal energy or temperature values as needed. The interpolation is done with the *linear* style. For the *linear* style, the internal temperature is used to find 2 surrounding table values from which an internal energy is computed by linear interpolation. A secant solver is used to determine the internal temperature from the internal energy.

The first filename specifies a file containing tabulated internal temperature and m internal energy values for each species u_j . The keyword specifies a section of the file. The format of this file is described below.

The second filename specifies a file containing heat of formation $\Delta H_{f,j}$ for each species.

In cases where the coarse-grain particle represents a single molecular species (i.e., no reactions occur and fix *rx* is not present in the input file), fix *eos/table/rx* can be applied in a similar manner to fix *eos/table* within a non-reactive DPD simulation. In this case, the heat of formation filename is replaced with the heat of formation value for the single species. Additionally, the energy correction and temperature correction coefficients may also be specified as fix arguments.

The format of a tabulated file is as follows (without the parenthesized comments):

# EOS TABLE	(one or more comment or blank lines)
KEYWORD	(keyword is first text on line)
N 500 h2 n2 ... no	(N parameter species1 species2 ... speciesN) (blank)
1 1.00 0.000 ... 0.0000	(index, internal temperature, internal energy of species 1, ..., internal energy of species m)
2 1.02 0.001 ... 0.0002	
...	
500 10.0 0.500 ... 1.0000	

A section begins with a non-blank line whose first character is not a “#”; blank lines or lines starting with “#” can be used as comments between sections. The first line begins with a keyword which identifies the section. The line can contain additional text, but the initial text must match the argument specified in the fix command.

The next line lists the number of table entries and the species names that correspond with all the species listed in the reaction equations through the *fix rx* command. The parameter “N” is required and its value is the number of table entries that follow. Let Nfile = “N” in the tabulated file. What LAMMPS does is a preliminary interpolation by creating splines using the Nfile tabulated values as nodal points.

Following a blank line, the next N lines list the tabulated values. On each line, the first value is the index from 1 to N, the second value is the internal temperature (in temperature units), the third value until the *m+3* value are the internal energies of the *m* species (in energy units).

Note that all internal temperature and internal energy values must increase from one line to the next.

Note that one file can contain many sections, each with a tabulated potential. LAMMPS reads the file section by section until it finds one that matches the specified keyword.

The format of a heat of formation file is as follows (without the parenthesized comments):

```
# HEAT OF FORMATION TABLE (one or more comment or blank lines)

                                (blank)
h2      0.00          (species name, heat of formation)
no2     0.34
n2      0.00
...
no      0.93
```

Note that the species can be listed in any order. The tag that is used as the species name must correspond with the tags used to define the reactions with the *fix rx* command.

Alternatively, corrections to the EOS can be included by specifying three additional columns that correspond to the energy correction, the temperature correction coefficient and molecule correction coefficient. In this case, the format of the file is as follows:

```
# HEAT OF FORMATION TABLE (one or more comment or blank lines)

                                (blank)
h2      0.00 1.23 0.025 0.0 (species name, heat of formation, energy correction, temperature correction
    ↗ coefficient, molecule correction coefficient)
no2     0.34 0.00 0.000 -1.76
n2      0.00 0.00 0.000 -1.76
...
no      0.93 0.00 0.000 -1.76
```

Styles with a *gpu*, *intel*, *kk*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed on the [Accelerator packages](#) page. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the GPU, INTEL, KOKKOS, OPENMP, and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the [Build package](#) page for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the *-suffix command-line switch* when you invoke LAMMPS, or you can use the *suffix* command in your input script.

See the [Accelerator packages](#) page for more instructions on how to use the accelerated styles effectively.

2.63.4 Restrictions

This command is part of the DPD-REACT package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

This command also requires use of the [atom_style dpd](#) command.

The equation of state must be a monotonically increasing function.

An error will occur if the internal temperature or internal energies are not within the table cutoffs.

2.63.5 Related commands

[fix rx](#), [pair dpd/fdt](#)

2.63.6 Default

none

2.64 fix evaporate command

2.64.1 Syntax

```
fix ID group-ID evaporate N M region-ID seed
```

- ID, group-ID are documented in [fix](#) command
- evaporate = style name of this fix command
- N = delete atoms every this many timesteps
- M = number of atoms to delete each time
- region-ID = ID of region within which to perform deletions
- seed = random number seed to use for choosing atoms to delete
- zero or more keyword/value pairs may be appended

keyword = molecule
molecule value = no or yes

2.64.2 Examples

```
fix 1 solvent evaporate 1000 10 surface 49892
fix 1 solvent evaporate 1000 10 surface 38277 molecule yes
```

2.64.3 Description

Remove M atoms from the simulation every N steps. This can be used, for example, to model evaporation of solvent particles or molecules (i.e. drying) of a system. Every N steps, the number of atoms in the fix group and within the specified region are counted. M of these are chosen at random and deleted. If there are less than M eligible particles, then all of them are deleted.

If the setting for the *molecule* keyword is *no*, then only single atoms are deleted. In this case, you should ensure you do not delete only a portion of a molecule (only some of its atoms), or LAMMPS will soon generate an error when it tries to find those atoms. LAMMPS will warn you if any of the atoms eligible for deletion have a non-zero molecule ID, but does not check for this at the time of deletion.

If the setting for the *molecule* keyword is *yes*, then when an atom is chosen for deletion, the entire molecule it is part of is deleted. The count of deleted atoms is incremented by the number of atoms in the molecule, which may make it exceed *M*. If the molecule ID of the chosen atom is 0, then it is assumed to not be part of a molecule, and just the single atom is deleted.

As an example, if you wish to delete 10 water molecules every *N* steps, you should set *M* to 30. If only the water's oxygen atoms were in the fix group, then two hydrogen atoms would be deleted when an oxygen atom is selected for deletion, whether the hydrogen atoms are inside the evaporation region or not.

Note that neighbor lists are re-built on timesteps that atoms are removed. Thus you should not remove atoms too frequently or you will incur overhead due to the cost of building neighbor lists.

Note

If you are monitoring the temperature of a system where the atom count is changing due to evaporation, you typically should use the *compute_modify dynamic/dof yes* command for the temperature compute you are using.

2.64.4 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 cumulative number of deleted atoms. The scalar value calculated by this fix is “intensive”.

No parameter of this fix can be used with the *start/stop* keywords of the *run* command. This fix is not invoked during *energy minimization*.

2.64.5 Restrictions

None

2.64.6 Related commands

fix deposit

2.64.7 Default

The option defaults are molecule = no.

2.65 fix external command

2.65.1 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

2.65.2 Examples

```
fix 1 all external pf/callback 1 1
fix 1 all external pf/callback 100 1
fix 1 all external pf/array 10
```

2.65.3 Description

This fix allows external programs that are running LAMMPS through its *library interface* to modify certain LAMMPS 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, tagint *ids, double **x, double **fexternal);
```

The arguments are as follows:

- *ptr* = pointer provided by and simply passed back to external driver
- *timestep* = current LAMMPS timestep
- *nlocal* = # of atoms on this processor
- *ids* = list of atom IDs on this processor
- *x* = coordinates of atoms on this processor
- *fexternal* = forces to add to atoms on this processor

Note that *timestep* is a “bigint” which is defined in src/lmp.h, typically as a 64-bit integer. And *ids* is a pointer to type “tagint” which is typically a 32-bit integer unless LAMMPS is compiled with -DLAMMPS_BIGBIG. For more info please see the [build settings](#) section of the manual. Finally, *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 LAMMPS 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 LAMMPS [run](#) command, it could retrieve atom coordinates from LAMMPS, 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. Correspondingly, the global virial needs to be used this fix with variable cell calculations (e.g. [fix box/relax](#) or [fix npt](#)).

This can be done from the external driver by calling these public methods of the FixExternal class:

```
void set_energy_global(double eng);
void set_virial_global(double *virial);
```

where *eng* is the potential energy, and *virial* an array of the 6 stress tensor components. 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 ensure this energy setting is used appropriately in a minimization.

Additional public methods that the caller can use to update system properties are:

```
void set_energy_peratom(double *eng);
void set_virial_peratom(double **virial);
void set_vector_length(int n);
void set_vector(int idx, double val);
```

These allow to set per-atom energy contributions, per-atom stress contributions, the length and individual values of a global vector of properties that the caller code may want to communicate to LAMMPS (e.g. for use in [fix ave/time](#) or in [equal-style variables](#) or for [custom thermo output](#)).

2.65.4 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 both the global potential energy and peratom potential energies of the system as part of [thermodynamic output](#) or output by the [compute pe/atom](#) command. The default setting for this fix is [fix_modify energy yes](#). Note that this energy may be a fictitious quantity but it 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.

The [fix_modify virial](#) option is supported by this fix to add the contribution computed by the external program to both the global pressure and per-atom stress of the system via the [compute pressure](#) and [compute stress/atom](#) commands. The former can be accessed by [thermodynamic output](#). The default setting for this fix is [fix_modify virial yes](#).

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.

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 not disable the [fix_modify energy](#) option for this fix.

2.65.5 Restrictions

none

2.65.6 Related commands

none

2.65.7 Default

none

2.66 fix ffl command

2.66.1 Syntax

```
fix ID id-group ffl tau Tstart Tstop seed [flip-type]
```

- ID, group-ID are documented in [fix](#) command
- ffl = style name of this fix command
- tau = thermostat parameter (positive real)
- Tstart, Tstop = temperature ramp during the run
- seed = random number seed to use for generating noise (positive integer)
- one more value may be appended

flip-type = determines the flipping type, can be chosen between rescale - no_flip - hard - soft, if no_flip type is given, rescale will be chosen by default

2.66.2 Examples

```
fix 3 boundary ffl 10 300 300 31415
fix 1 all ffl 100 500 500 9265 soft
```

2.66.3 Description

Apply a Fast-Forward Langevin Equation (FFL) thermostat as described in ([Hijazi](#)). Contrary to [fix langevin](#), this fix performs both thermostating and evolution of the Hamiltonian equations of motion, so it should not be used together with [fix nve](#) – at least not on the same atom groups.

The time-evolution of a single particle undergoing Langevin dynamics is described by the equations

$$\frac{dq}{dt} = \frac{p}{m},$$

$$\frac{dp}{dt} = -\gamma p + W + F,$$

where F is the physical force, γ is the friction coefficient, and W is a Gaussian random force.

The friction coefficient is the inverse of the thermostat parameter : $\gamma = 1/\tau$, with τ the thermostat parameter *tau*. The thermostat parameter is given in the time units, γ is in inverse time units.

Equilibrium sampling a temperature T is obtained by specifying the target value as the *Tstart* and *Tstop* arguments, so that the internal constants depending on the temperature are computed automatically.

The random number *seed* must be a positive integer. A Marsaglia random number generator is used. Each processor uses the input seed to generate its own unique seed and its own stream of random numbers. Thus the dynamics of the system will not be identical on two runs on different numbers of processors.

The flipping type *flip-type* can be chosen between 4 types described in ([Hijazi](#)). The flipping operation occurs during the thermostating step and it flips the momenta of the atoms. If no_flip is chosen, no flip will be executed and the integration will be the same as a standard Langevin thermostat ([Bussi](#)). The other flipping types are : rescale - hard - soft.

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

The instantaneous values of the extended variables are written to *binary restart files*. Because the state of the random number generator is not saved in restart files, this means you cannot do “exact” restarts with this fix, where the simulation continues on the same as if no restart had taken place. However, in a statistical sense, a restarted simulation should produce the same behavior. Note however that you should use a different seed each time you restart, otherwise the same sequence of random numbers will be used each time, which might lead to stochastic synchronization and subtle artifacts in the sampling.

The cumulative energy change in the system imposed by this fix is included in the *thermodynamic output* keywords *ecouple* and *econserve*. See the *thermo_style* doc page for details.

This fix computes a global scalar which can be accessed by various *output commands*. The scalar is the same cumulative energy change due to this fix described in the previous paragraph. The scalar value calculated by this fix is “extensive”.

This fix can ramp its target temperature over multiple runs, using the *start* and *stop* keywords of the *run* command. See the *run* command for details of how to do this.

This fix is not invoked during *energy minimization*.

2.66.5 Restrictions

In order to perform constant-pressure simulations please use *fix press/berendsen*, rather than *fix npt*, to avoid duplicate integration of the equations of motion.

This fix is part of the EXTRA-FIX package. It is only enabled if LAMMPS was built with that package. See the *Build package* page for more info.

2.66.6 Related commands

fix nvt, *fix temp/rescale*, *fix viscous*, *fix nvt*, *pair_style dpd/tstat*, *fix gld*, *fix gle*

(Hijazi) M. Hijazi, D. M. Wilkins, M. Ceriotti, J. Chem. Phys. 148, 184109 (2018)

(Bussi) G. Bussi, M. Parrinello, Phs. Rev. E 75, 056707 (2007)

2.67 fix filter/corotate command

2.67.1 Syntax

```
fix ID group-ID filter/corotate keyword value ...
```

- ID, group-ID are documented in *fix* command
- one or more constraint/value pairs are appended
- constraint = *b* or *a* or *t* or *m*

b values = one or more bond types
a values = one or more angle types
t values = one or more atom types
m value = one or more mass values

2.67.2 Examples

```
timestep 8
run_style respa 3 2 8 bond 1 pair 2 kspace 3
fix cor all filter/corotate m 1.0

fix cor all filter/corotate b 4 19 a 3 5 2
```

2.67.3 Description

This fix implements a corotational filter for a mollified impulse method. In biomolecular simulations, it allows the usage of larger timesteps for long-range electrostatic interactions. For details, see ([Fath](#)).

When using [run_style respa](#) for a biomolecular simulation with high-frequency covalent bonds, the outer time-step is restricted to below ~ 4fs due to resonance problems. This fix filters the outer stage of the respa and thus a larger (outer) time-step can be used. Since in large biomolecular simulations the computation of the long-range electrostatic contributions poses a major bottleneck, this can significantly accelerate the simulation.

The filter computes a cluster decomposition of the molecular structure following the criteria indicated by the options a, b, t and m. This process is similar to the approach in [fix shake](#), however, the clusters are not kept constrained. Instead, the position is slightly modified only for the computation of long-range forces. A good cluster decomposition constitutes in building clusters which contain the fastest covalent bonds inside clusters.

If the clusters are chosen suitably, the [run_style respa](#) is stable for outer timesteps of at least 8fs.

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

No information about these fixes is written to [binary restart files](#). None of the [fix_modify](#) options are relevant to these fixes. No global or per-atom quantities are stored by these fixes for access by various [output commands](#). 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](#).

2.67.5 Restrictions

This fix is part of the EXTRA-FIX package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

Currently, it does not support [molecule templates](#).

2.67.6 Related commands

2.67.7 Default

none

(**Fath**) Fath, Hochbruck, Singh, J Comp Phys, 333, 180-198 (2017).

2.68 fix flow/gauss command

2.68.1 Syntax

```
fix ID group-ID flow/gauss xflag yflag zflag keyword
```

- ID, group-ID are documented in [fix](#) command
- flow/gauss = style name of this fix command
- xflag,yflag,zflag = 0 or 1

0 = do not conserve current in this dimension
 1 = conserve current in this dimension

- zero or more keyword/value pairs may be appended
- keyword = *energy*
 energy value = no or yes
 no = do not compute work done by this fix
 yes = compute work done by this fix

2.68.2 Examples

```
fix GD fluid flow/gauss 1 0 0
fix GD fluid flow/gauss 1 1 1 energy yes
```

2.68.3 Description

This fix implements the Gaussian dynamics (GD) method to simulate a system at constant mass flux ([Strong](#)). GD is a nonequilibrium molecular dynamics simulation method that can be used to study fluid flows through pores, pipes, and channels. In its original implementation GD was used to compute the pressure required to achieve a fixed mass flux through an opening. The flux can be conserved in any combination of the directions, x, y, or z, using xflag,yflag,zflag. This fix does not initialize a net flux through a system, it only conserves the center-of-mass momentum that is present when the fix is declared in the input script. Use the [velocity](#) command to generate an initial center-of-mass momentum.

GD applies an external fluctuating gravitational field that acts as a driving force to keep the system away from equilibrium. To maintain steady state, a profile-unbiased thermostat must be implemented to dissipate the heat that is added by the driving force. [Compute temp/profile](#) can be used to implement a profile-unbiased thermostat.

A common use of this fix is to compute a pressure drop across a pipe, pore, or membrane. The pressure profile can be computed in LAMMPS with [compute stress/atom](#) and [fix ave/chunk](#), or with the hardy method in [fix atc](#). Note that the simple [compute stress/atom](#) method is only accurate away from inhomogeneities in the fluid, such as fixed wall atoms. Further, the computed pressure profile must be corrected for the acceleration applied by GD before computing a pressure drop or comparing it to other methods, such as the pump method ([Zhu](#)). The pressure correction is discussed and described in ([Strong](#)).

For a complete example including the considerations discussed above, see the examples/PACKAGES/flow_gauss directory.

Note

Only the flux of the atoms in group-ID will be conserved. If the velocities of the group-ID atoms are coupled to the velocities of other atoms in the simulation, the flux will not be conserved. For example, in a simulation with fluid atoms and harmonically constrained wall atoms, if a single thermostat is applied to group *all*, the fluid atom velocities will be coupled to the wall atom velocities, and the flux will not be conserved. This issue can be avoided by thermostating the fluid and wall groups separately.

Adding an acceleration to atoms does work on the system. This added energy can be optionally subtracted from the potential energy for the thermodynamic output (see below) to check that the timestep is small enough to conserve energy. Since the applied acceleration is fluctuating in time, the work cannot be computed from a potential. As a result, computing the work is slightly more computationally expensive than usual, so it is not performed by default. To invoke the work calculation, use the *energy* keyword. The *fix_modify energy* option also invokes the work calculation, and overrides an *energy no* setting here. If neither *energy yes* or *fix_modify energy yes* are set, the global scalar computed by the fix will return zero.

Note

In order to check energy conservation, any other fixes that do work on the system must have *fix_modify energy yes* set as well. This includes thermostat fixes and any constraints that hold the positions of wall atoms fixed, such as *fix spring/self*.

If this fix is used in a simulation with the *rRESPA* integrator, the applied acceleration must be computed and applied at the same rRESPA level as the interactions between the flowing fluid and the obstacle. The rRESPA level at which the acceleration is applied can be changed using the *fix_modify respa* option discussed below. If the flowing fluid and the obstacle interact through multiple interactions that are computed at different rRESPA levels, then there must be a separate flow/gauss fix for each level. For example, if the flowing fluid and obstacle interact through pairwise and long-range Coulomb interactions, which are computed at rRESPA levels 3 and 4, respectively, then there must be two separate flow/gauss fixes, one that specifies *fix_modify respa 3* and one with *fix_modify respa 4*.

2.68.4 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 added by the fix to the global potential energy of the system as part of *thermodynamic output*. The default setting for this fix is *fix_modify energy no*.

The *fix_modify respa* option is supported by this fix. This allows the user to set at which level of the *rRESPA* integrator the fix computes and adds the external acceleration. Default is the outermost level.

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 negative of the work done on the system, see the discussion above. It is only calculated if the *energy* keyword is enabled or *fix_modify energy yes* is set.

The vector is the total force that this fix applied to the group of atoms on the current timestep. 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.

This fix is not invoked during *energy minimization*.

2.68.5 Restrictions

This fix is part of the EXTRA-FIX package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

2.68.6 Related commands

fix addforce, compute temp/profile, velocity

2.68.7 Default

The option default for the *energy* keyword is *energy = no*.

(Strong) Strong and Eaves, J. Phys. Chem. B 121, 189 (2017).

(Evans) Evans and Morriss, Phys. Rev. Lett. 56, 2172 (1986).

(Zhu) Zhu, Tajkhorshid, and Schulten, Biophys. J. 83, 154 (2002).

2.69 fix freeze command

Accelerator Variants: *freeze/kk*

2.69.1 Syntax

```
fix ID group-ID freeze
```

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

2.69.2 Examples

```
fix 2 bottom freeze
```

2.69.3 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. A similar functionality for normal (point) particles can be obtained using [fix setforce](#).

Styles with a *gpu*, *intel*, *kk*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed on the [Accelerator packages](#) page. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the GPU, INTEL, KOKKOS, OPENMP, and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the [Build package](#) page for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the [-suffix command-line switch](#) when you invoke LAMMPS, or you can use the [suffix](#) command in your input script.

See the [Accelerator packages](#) page for more instructions on how to use the accelerated styles effectively.

2.69.4 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](#).

2.69.5 Restrictions

This fix is part of the GRANULAR package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

There can only be a single freeze fix defined. This is because other the [granular pair styles](#) treat frozen particles differently and need to be able to reference a single group to which this fix is applied.

2.69.6 Related commands

[atom_style sphere](#), [fix setforce](#)

2.69.7 Default

none

2.70 fix gcmc command

2.70.1 Syntax

```
fix ID group-ID gcmc N X M type seed T mu displace keyword values ...
```

- ID, group-ID are documented in [fix](#) command
- gcmc = style name of this fix command
- N = invoke this fix every N steps
- X = average number of GCMC exchanges to attempt every N steps
- M = average number of MC moves to attempt every N steps
- type = atom type (1-Ntypes or type label) for inserted atoms (must be 0 if mol keyword used)

- seed = random # seed (positive integer)
 - T = temperature of the ideal gas reservoir (temperature units)
 - mu = chemical potential of the ideal gas reservoir (energy units)
 - displace = maximum Monte Carlo translation distance (length units)
 - zero or more keyword/value pairs may be appended to args
- keyword = mol, region, maxangle, pressure, fugacity_coeff, full_energy, charge, group, grouptype,
↳ intra_energy, tfac_insert, or overlap_cutoff
- mol value = template-ID
 template-ID = ID of molecule template specified in a separate molecule command
- mcmoves values = Patomtrans Pmoltrans Pmolrotate
 Patomtrans = proportion of atom translation MC moves
 Pmoltrans = proportion of molecule translation MC moves
 Pmolrotate = proportion of molecule rotation MC moves
- rigid value = fix-ID
 fix-ID = ID of fix rigid/small command
- shake value = fix-ID
 fix-ID = ID of fix shake command
- region value = region-ID
 region-ID = ID of region where GCMC exchanges and MC moves are allowed
- maxangle value = maximum molecular rotation angle (degrees)
- pressure value = pressure of the gas reservoir (pressure units)
- fugacity_coeff value = fugacity coefficient of the gas reservoir (unitless)
- full_energy = compute the entire system energy when performing GCMC exchanges and MC
↳ moves
- charge value = charge of inserted atoms (charge units)
- group value = group-ID
 group-ID = group-ID for inserted atoms (string)
- grouptype values = type group-ID
 type = atom type (1-Ntypes or type label)
 group-ID = group-ID for inserted atoms (string)
- intra_energy value = intramolecular energy (energy units)
- tfac_insert value = scale up/down temperature of inserted atoms (unitless)
- overlap_cutoff value = maximum pair distance for overlap rejection (distance units)
- max value = Maximum number of molecules allowed in the system
- min value = Minimum number of molecules allowed in the system

2.70.2 Examples

```
fix 2 gas gcmc 10 1000 1000 2 29494 298.0 -0.5 0.01
fix 3 water gcmc 10 100 100 0 3456543 3.0 -2.5 0.1 mol my_one_water maxangle 180 full_energy
fix 4 my_gas gcmc 1 10 10 1 123456543 300.0 -12.5 1.0 region disk

labelmap atom 1 Li
fix 2 ion gcmc 10 1000 1000 Li 29494 298.0 -0.5 0.01
```

2.70.3 Description

This fix performs grand canonical Monte Carlo (GCMC) exchanges of atoms or molecules with an imaginary ideal gas reservoir at the specified T and chemical potential (mu) as discussed in ([Frenkel](#)). It also attempts Monte Carlo (MC) moves (translations and molecule rotations) within the simulation cell or region. If used with the [fix nvt](#) command, simulations in the grand canonical ensemble (muVT, constant chemical potential, constant volume, and constant temperature) can be performed. Specific uses include computing isotherms in microporous materials, or computing vapor-liquid coexistence curves.

Every N timesteps the fix attempts both GCMC exchanges (insertions or deletions) and MC moves of gas atoms or molecules. On those timesteps, the average number of attempted GCMC exchanges is X, while the average number of attempted MC moves is M. For GCMC exchanges of either molecular or atomic gasses, these exchanges can be either deletions or insertions, with equal probability.

The possible choices for MC moves are translation of an atom, translation of a molecule, and rotation of a molecule. The relative amounts of each are determined by the optional *mcmoves* keyword (see below). The default behavior is as follows. If the *mol* keyword is used, only molecule translations and molecule rotations are performed with equal probability. Conversely, if the *mol* keyword is not used, only atom translations are performed. M should typically be chosen to be approximately equal to the expected number of gas atoms or molecules of the given type within the simulation cell or region, which will result in roughly one MC move per atom or molecule per MC cycle.

All inserted particles are always added to two groups: the default group “all” and the fix group specified in the fix command. In addition, particles are also added to any groups specified by the *group* and *grouptype* keywords. If inserted particles are individual atoms, they are assigned the atom type given by the *type* argument. If they are molecules, the *type* argument has no effect and must be set to zero. Instead, the type of each atom in the inserted molecule is specified in the file read by the [molecule](#) command.

Note

Care should be taken to apply fix gcmc only to a group that contains only those atoms and molecules that you wish to manipulate using Monte Carlo. Hence it is generally not a good idea to specify the default group “all” in the fix command, although it is allowed.

This fix cannot be used to perform GCMC insertions of gas atoms or molecules other than the exchanged type, but GCMC deletions, and MC translations, and rotations can be performed on any atom/molecule in the fix group. All atoms in the simulation cell can be moved using regular time integration translations, e.g. via [fix nvt](#), resulting in a hybrid GCMC+MD simulation. A smaller-than-usual timestep size may be needed when running such a hybrid simulation, especially if the inserted molecules are not well equilibrated.

This command may optionally use the *region* keyword to define an exchange and move volume. The specified region must have been previously defined with a [region](#) command. It must be defined with side = *in*. Insertion attempts occur only within the specified region. For non-rectangular regions, random trial points are generated within the rectangular bounding box until a point is found that lies inside the region. If no valid point is generated after 1000 trials, no insertion is performed, but it is counted as an attempted insertion. Move and deletion attempt candidates are selected from gas atoms or molecules within the region. If there are no candidates, no move or deletion is performed, but it is counted as an attempt move or deletion. If an attempted move places the atom or molecule center-of-mass outside the specified region, a new attempted move is generated. This process is repeated until the atom or molecule center-of-mass is inside the specified region.

If used with [fix nvt](#), the temperature of the imaginary reservoir, T, should be set to be equivalent to the target temperature used in fix nvt. Otherwise, the imaginary reservoir will not be in thermal equilibrium with the simulation cell. Also, it is important that the temperature used by [fix nvt](#) is dynamically updated, which can be achieved as follows:

```
compute mdtemp mdatoms temp
compute _modify mdtemp dynamic/dof yes
```

(continues on next page)

(continued from previous page)

```
fix mdnvt mdatoms nvt temp 300.0 300.0 10.0
fix_modify mdnvt temp mdtemp
```

Note that neighbor lists are re-built every timestep that this fix is invoked, so you should not set N to be too small. However, periodic rebuilds are necessary in order to avoid dangerous rebuilds and missed interactions. Specifically, avoid performing so many MC translations per timestep that atoms can move beyond the neighbor list skin distance. See the [neighbor](#) command for details.

When an atom or molecule is to be inserted, its coordinates are chosen at a random position within the current simulation cell or region, and new atom velocities are randomly chosen from the specified temperature distribution given by T. The effective temperature for new atom velocities can be increased or decreased using the optional keyword *tfac_insert* (see below). Relative coordinates for atoms in a molecule are taken from the template molecule provided by the user. The center of mass of the molecule is placed at the insertion point. The orientation of the molecule is chosen at random by rotating about this point.

Individual atoms are inserted, unless the *mol* keyword is used. It specifies a *template-ID* previously defined using the [molecule](#) command, which reads a file that defines the molecule. The coordinates, atom types, charges, etc., as well as any bonding and special neighbor information for the molecule can be specified in the molecule file. See the [molecule](#) command for details. The only settings required to be in this file are the coordinates and types of atoms in the molecule.

When not using the *mol* keyword, you should ensure you do not delete atoms that are bonded to other atoms, or LAMMPS will soon generate an error when it tries to find bonded neighbors. LAMMPS will warn you if any of the atoms eligible for deletion have a non-zero molecule ID, but does not check for this at the time of deletion.

If you wish to insert molecules using the *mol* keyword that will be treated as rigid bodies, use the *rigid* keyword, specifying as its value the ID of a separate [fix rigid/small](#) command which also appears in your input script.

Note

If you wish the new rigid molecules (and other rigid molecules) to be thermostatted correctly via [fix rigid/small/nvt](#) or [fix rigid/small/npt](#), then you need to use the [fix_modify dynamic/dof yes](#) command for the rigid fix. This is to inform that fix that the molecule count will vary dynamically.

If you wish to insert molecules via the *mol* keyword, that will have their bonds or angles constrained via SHAKE, use the *shake* keyword, specifying as its value the ID of a separate [fix shake](#) command which also appears in your input script.

Optionally, users may specify the relative amounts of different MC moves using the *mcmoves* keyword. The values *Patomtrans*, *Pmoltrans*, *Pmolrotate* specify the average proportion of atom translations, molecule translations, and molecule rotations, respectively. The values must be non-negative integers or real numbers, with at least one non-zero value. For example, (10,30,0) would result in 25% of the MC moves being atomic translations, 75% molecular translations, and no molecular rotations.

Optionally, users may specify the maximum rotation angle for molecular rotations using the *maxangle* keyword and specifying the angle in degrees. Rotations are performed by generating a random point on the unit sphere and a random rotation angle on the range [0,maxangle]. The molecule is then rotated by that angle about an axis passing through the molecule center of mass. The axis is parallel to the unit vector defined by the point on the unit sphere. The same procedure is used for randomly rotating molecules when they are inserted, except that the maximum angle is 360 degrees.

Note that fix gcmc does not use configurational bias MC or any other kind of sampling of intramolecular degrees of freedom. Inserted molecules can have different orientations, but they will all have the same intramolecular configuration, which was specified in the molecule command input.

For atomic gasses, inserted atoms have the specified atom type, but deleted atoms are any atoms that have been inserted or that already belong to the fix group. For molecular gasses, exchanged molecules use the same atom types as in the

template molecule supplied by the user. In both cases, exchanged atoms/molecules are assigned to two groups: the default group “all” and the fix group (which can also be “all”).

The chemical potential is a user-specified input parameter defined as:

$$\mu = \mu^{id} + \mu^{ex}$$

The second term `mu_ex` is the excess chemical potential due to energetic interactions and is formally zero for the fictitious gas reservoir but is non-zero for interacting systems. So, while the chemical potential of the reservoir and the simulation cell are equal, `mu_ex` is not, and as a result, the densities of the two are generally quite different. The first term `mu_id` is the ideal gas contribution to the chemical potential. `mu_id` can be related to the density or pressure of the fictitious gas reservoir by:

$$\begin{aligned}\mu^{id} &= kT \ln \rho \Lambda^3 \\ &= kT \ln \frac{\phi P \Lambda^3}{k_B T}\end{aligned}$$

where k_B is the Boltzmann constant, T is the user-specified temperature, ρ is the number density, P is the pressure, and ϕ is the fugacity coefficient. The constant Λ is required for dimensional consistency. For all unit styles except `lj` it is defined as the thermal de Broglie wavelength

$$\Lambda = \sqrt{\frac{h^2}{2\pi m k_B T}}$$

where h is Planck’s constant, and m is the mass of the exchanged atom or molecule. For unit style `lj`, Λ is simply set to unity. Note that prior to March 2017, Λ for unit style `lj` was calculated using the above formula with h set to the rather specific value of 0.18292026. Chemical potential under the old definition can be converted to an equivalent value under the new definition by subtracting $3kT \ln(\Lambda_{old})$.

As an alternative to specifying `mu` directly, the ideal gas reservoir can be defined by its pressure P using the `pressure` keyword, in which case the user-specified chemical potential is ignored. The user may also specify the fugacity coefficient ϕ using the `fugacity_coeff` keyword, which defaults to unity.

The `full_energy` option means that the fix calculates the total potential energy of the entire simulated system, instead of just the energy of the part that is changed. The total system energy before and after the proposed GCMC exchange or MC move is then used in the Metropolis criterion to determine whether or not to accept the proposed change. By default, this option is off, in which case only partial energies are computed to determine the energy difference due to the proposed change.

The `full_energy` option is needed for systems with complicated potential energy calculations, including the following:

- long-range electrostatics (`kspace`)
- many-body pair styles
- hybrid pair styles
- eam pair styles
- tail corrections
- need to include potential energy contributions from other fixes

In these cases, LAMMPS will automatically apply the `full_energy` keyword and issue a warning message.

When the `mol` keyword is used, the `full_energy` option also includes the intramolecular energy of inserted and deleted molecules, whereas this energy is not included when `full_energy` is not used. If this is not desired, the `intra_energy` keyword can be used to define an amount of energy that is subtracted from the final energy when a molecule is inserted, and subtracted from the initial energy when a molecule is deleted. For molecules that have a non-zero intramolecular energy, this will ensure roughly the same behavior whether or not the `full_energy` option is used.

Inserted atoms and molecules are assigned random velocities based on the specified temperature T . Because the relative velocity of all atoms in the molecule is zero, this may result in inserted molecules that are systematically too cold. In addition, the intramolecular potential energy of the inserted molecule may cause the kinetic energy of the molecule to quickly increase or decrease after insertion. The *tfac_insert* keyword allows the user to counteract these effects by changing the temperature used to assign velocities to inserted atoms and molecules by a constant factor. For a particular application, some experimentation may be required to find a value of *tfac_insert* that results in inserted molecules that equilibrate quickly to the correct temperature.

Some fixes have an associated potential energy. Examples of such fixes include: *field*, *gravity*, *addforce*, *langevin*, *restrain*, *temp/berendsen*, *temp/rescale*, and *wall fixes*. For that energy to be included in the total potential energy of the system (the quantity used when performing GCMC exchange and MC moves), you MUST enable the *fix_modify energy* option for that fix. The doc pages for individual *fix* commands specify if this should be done.

Use the *charge* option to insert atoms with a user-specified point charge. Note that doing so will cause the system to become non-neutral. LAMMPS issues a warning when using long-range electrostatics (kspace) with non-neutral systems. See the *compute group/group* documentation for more details about simulating non-neutral systems with kspace on.

Use of this fix typically will cause the number of atoms to fluctuate, therefore, you will want to use the *compute_modify dynamic/dof* command to ensure that the current number of atoms is used as a normalizing factor each time temperature is computed. A simple example of this is:

```
compute_modify thermo_temp dynamic/dof yes
```

A more complicated example is listed earlier on this page in the context of NVT dynamics.

Note

If the density of the cell is initially very small or zero, and increases to a much larger density after a period of equilibration, then certain quantities that are only calculated once at the start (kspace parameters) may no longer be accurate. The solution is to start a new simulation after the equilibrium density has been reached.

With some pair_styles, such as *Buckingham*, *Born-Mayer-Huggins* and *ReaxFF*, two atoms placed close to each other may have an arbitrary large, negative potential energy due to the functional form of the potential. While these unphysical configurations are inaccessible to typical dynamical trajectories, they can be generated by Monte Carlo moves. The *overlap_cutoff* keyword suppresses these moves by effectively assigning an infinite positive energy to all new configurations that place any pair of atoms closer than the specified overlap cutoff distance.

The *max* and *min* keywords allow for the restriction of the number of atoms in the simulation. They automatically reject all insertion or deletion moves that would take the system beyond the set boundaries. Should the system already be beyond the boundary, only moves that bring the system closer to the bounds may be accepted.

The *group* keyword adds all inserted atoms to the *group* of the group-ID value. The *group/type* keyword adds all inserted atoms of the specified type to the *group* of the group-ID value.

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

This fix writes the state of the fix to *binary restart files*. This includes information about the random number generator seed, the next timestep for MC exchanges, the number of MC step attempts and successes etc. See the *read_restart* command for info on how to re-specify a fix in an input script that reads a restart file, so that the operation of the fix continues in an uninterrupted fashion.

Note

For this to work correctly, the timestep must **not** be changed after reading the restart with `reset_timestep`. The fix will try to detect it and stop with an error.

None of the `fix_modify` options are relevant to this fix.

This fix computes a global vector of length 8, which can be accessed by various *output commands*. The vector values are the following global cumulative quantities:

1. translation attempts
2. translation successes
3. insertion attempts
4. insertion successes
5. deletion attempts
6. deletion successes
7. rotation attempts
8. rotation successes

The vector values calculated by this fix are “intensive”.

No parameter of this fix can be used with the *start/stop* keywords of the `run` command. This fix is not invoked during *energy minimization*.

2.70.5 Restrictions

This fix is part of the MC package. It is only enabled if LAMMPS was built with that package. See the *Build package* doc page for more info.

This fix style requires an *atom style* with per atom type masses.

Do not set “neigh_modify once yes” or else this fix will never be called. Renighboring is **required**.

Only usable for 3D simulations.

This fix can be run in parallel, but aspects of the GCMC part will not scale well in parallel. Currently, molecule translations and rotations are not supported with more than one MPI process. It is still possible to do parallel molecule exchange without translation and rotation moves by setting MC moves to zero and/or by using the `mcmoves` keyword with `Pmoltrans = Pmolrotate = 0`.

When using fix gcmc in combination with fix shake or fix rigid, only GCMC exchange moves are supported, so the argument *M* must be zero.

When using fix gcmc in combination with fix rigid, deletion of the last remaining molecule is not allowed for technical reasons, and so the molecule count will never drop below 1, regardless of the specified chemical potential.

Note that very lengthy simulations involving insertions/deletions of billions of gas molecules may run out of atom or molecule IDs and trigger an error, so it is better to run multiple shorter-duration simulations. Likewise, very large molecules have not been tested and may turn out to be problematic.

Use of multiple fix gcmc commands in the same input script can be problematic if using a template molecule. The issue is that the user-referenced template molecule in the second fix gcmc command may no longer exist since it might have been deleted by the first fix gcmc command. An existing template molecule will need to be referenced by the user for each subsequent fix gcmc command.

2.70.6 Related commands

fix atom/swap, fix nvt, neighbor, fix deposit, fix evaporate, delete_atoms

2.70.7 Default

The option defaults are mol = no, maxangle = 10, overlap_cutoff = 0.0, fugacity_coeff = 1.0, intra_energy = 0.0, tfac_insert = 1.0. (Patomtrans, Pmoltrans, Pmolrotate) = (1, 0, 0) for mol = no and (0, 1, 1) for mol = yes. full_energy = no, except for the situations where full_energy is required, as listed above.

(Frenkel) Frenkel and Smit, Understanding Molecular Simulation, Academic Press, London, 2002.

2.71 fix gld command

2.71.1 Syntax

```
fix ID group-ID gld Tstart Tstop N_k seed series c_1 tau_1 ... c_N_k tau_N_k keyword values ...
```

- ID, group-ID are documented in [fix](#) command
 - gld = style name of this fix command
 - Tstart,Tstop = desired temperature at start/end of run (temperature units)
 - N_k = number of terms in the Prony series representation of the memory kernel
 - seed = random number seed to use for white noise (positive integer)
 - series = *pprony* is presently the only available option
 - c_k = the weight of the kth term in the Prony series (mass per time units)
 - tau_k = the time constant of the kth term in the Prony series (time units)
 - zero or more keyword/value pairs may be appended
- keyword = frozen or zero
 frozen value = no or yes
 no = initialize extended variables using values drawn from equilibrium distribution at Tstart
 yes = initialize extended variables to zero (i.e., from equilibrium distribution at zero temperature)
 zero value = no or yes
 no = do not set total random force to zero
 yes = set total random force to zero

2.71.2 Examples

```
fix 1 all gld 1.0 1.0 2 82885 ppronry 0.5 1.0 1.0 2.0 frozen yes zero yes
fix 3 rouse gld 7.355 7.355 4 48823 ppronry 107.1 0.02415 186.0 0.04294 428.6 0.09661 1714 0.38643
```

2.71.3 Description

Applies Generalized Langevin Dynamics to a group of atoms, as described in ([Baczewski](#)). This is intended to model the effect of an implicit solvent with a temporally non-local dissipative force and a colored Gaussian random force, consistent with the Fluctuation-Dissipation Theorem. The functional form of the memory kernel associated with the temporally non-local force is constrained to be a Prony series.

Note

While this fix bears many similarities to [fix langevin](#), it has one significant difference. Namely, [fix gld](#) performs time integration, whereas [fix langevin](#) does NOT. To this end, the specification of another fix to perform time integration, such as [fix nve](#), is NOT necessary.

With this fix active, the force on the j th atom is given as

$$\begin{aligned}\mathbf{F}_j(t) &= \mathbf{F}_j^C(t) - \int_0^t \Gamma_j(t-s) \mathbf{v}_j(s) \, ds + \mathbf{F}_j^R(t) \\ \Gamma_j(t-s) &= \sum_{k=1}^{N_k} \frac{c_k}{\tau_k} e^{-(t-s)/\tau_k} \\ \langle \mathbf{F}_j^R(t), \mathbf{F}_j^R(s) \rangle &= k_B T \Gamma_j(t-s)\end{aligned}$$

Here, the first term is representative of all conservative (pairwise, bonded, etc) forces external to this fix, the second is the temporally non-local dissipative force given as a Prony series, and the third is the colored Gaussian random force.

The Prony series form of the memory kernel is chosen to enable an extended variable formalism, with a number of exemplary mathematical features discussed in ([Baczewski](#)). In particular, $3N_k$ extended variables are added to each atom, which effect the action of the memory kernel without having to explicitly evaluate the integral over time in the second term of the force. This also has the benefit of requiring the generation of uncorrelated random forces, rather than correlated random forces as specified in the third term of the force.

Presently, the Prony series coefficients are limited to being greater than or equal to zero, and the time constants are limited to being greater than zero. To this end, the value of series MUST be set to `pprony`, for now. Future updates will allow for negative coefficients and other representations of the memory kernel. It is with these updates in mind that the series option was included.

The units of the Prony series coefficients are chosen to be mass per time to ensure that the numerical integration scheme stably approaches the Newtonian and Langevin limits. Details of these limits, and the associated numerical concerns are discussed in ([Baczewski](#)).

The desired temperature at each timestep is ramped from `Tstart` to `Tstop` over the course of the next run.

The random # `seed` must be a positive integer. A Marsaglia random number generator is used. Each processor uses the input seed to generate its own unique seed and its own stream of random numbers. Thus the dynamics of the system will not be identical on two runs on different numbers of processors.

The keyword/value option pairs are used in the following ways.

The keyword `frozen` can be used to specify how the extended variables associated with the GLD memory kernel are initialized. Specifying no (the default), the initial values are drawn at random from an equilibrium distribution at `Tstart`, consistent with the Fluctuation-Dissipation Theorem. Specifying yes, initializes the extended variables to zero.

The keyword `zero` can be used to eliminate drift due to the thermostat. Because the random forces on different atoms are independent, they do not sum exactly to zero. As a result, this fix applies a small random force to the entire system, and the center-of-mass of the system undergoes a slow random walk. If the keyword `zero` is set to yes, the total random force

is set exactly to zero by subtracting off an equal part of it from each atom in the group. As a result, the center-of-mass of a system with zero initial momentum will not drift over time.

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

The instantaneous values of the extended variables are written to *binary restart files*. Because the state of the random number generator is not saved in restart files, this means you cannot do “exact” restarts with this fix, where the simulation continues on the same as if no restart had taken place. However, in a statistical sense, a restarted simulation should produce the same behavior.

None of the *fix_modify* options are relevant to this fix. No global or per-atom quantities are stored by this fix for access by various *output commands*.

This fix can ramp its target temperature over multiple runs, using the *start* and *stop* keywords of the *run* command. See the *run* command for details of how to do this.

This fix is not invoked during *energy minimization*.

2.71.5 Restrictions

This fix is part of the EXTRA-FIX package. It is only enabled if LAMMPS was built with that package. See the *Build package* page for more info.

2.71.6 Related commands

fix langevin, fix viscous, pair_style dpd/tstat

2.71.7 Default

The option defaults are frozen = no, zero = no.

(Baczewski) A.D. Baczewski and S.D. Bond, J. Chem. Phys. 139, 044107 (2013).

2.72 fix gle command

2.72.1 Syntax

```
fix ID id-group gle Ns Tstart Tstop seed Amatrix [noneq Cmatrix] [every stride]
```

- ID, group-ID are documented in *fix* command
- gle = style name of this fix command
- Ns = number of additional fictitious momenta
- Tstart, Tstop = temperature ramp during the run
- Amatrix = file to read the drift matrix A from

- seed = random number seed to use for generating noise (positive integer)
- zero or more keyword/value pairs may be appended

keyword = noneq or every

noneq Cmatrix = file to read the non-equilibrium covariance matrix from
 every stride = apply the GLE once every time steps. Reduces the accuracy
 of the integration of the GLE, but has *no effect* on the accuracy of equilibrium
 sampling. It might change sampling properties when used together with noneq.

2.72.2 Examples

```
fix 3 boundary gle 6 300 300 31415 smart.A
fix 1 all gle 6 300 300 31415 qt-300k.A noneq qt-300k.C
```

2.72.3 Description

Apply a Generalized Langevin Equation (GLE) thermostat as described in ([Ceriotti](#)). The formalism allows one to obtain a number of different effects ranging from efficient sampling of all vibrational modes in the system to inexpensive (approximate) modelling of nuclear quantum effects. Contrary to [fix langevin](#), this fix performs both thermostating and evolution of the Hamiltonian equations of motion, so it should not be used together with [fix nve](#) – at least not on the same atom groups.

Each degree of freedom in the thermostatted group is supplemented with Ns additional degrees of freedom s, and the equations of motion become

```
dq/dt=p/m
d(p,s)/dt=(F,0) - A(p,s) + B dW/dt
```

where F is the physical force, A is the drift matrix (that generalizes the friction in Langevin dynamics), B is the diffusion term and dW/dt un-correlated Gaussian random forces. The A matrix couples the physical (q,p) dynamics with that of the additional degrees of freedom, and makes it possible to obtain effectively a history-dependent noise and friction kernel.

The drift matrix should be given as an external file *Afile*, as a (Ns+1 x Ns+1) matrix in inverse time units. Matrices that are optimal for a given application and the system of choice can be obtained from ([GLE4MD](#)).

Equilibrium sampling a temperature T is obtained by specifying the target value as the *Tstart* and *Tstop* arguments, so that the diffusion matrix that gives canonical sampling for a given A is computed automatically. However, the GLE framework also allow for non-equilibrium sampling, that can be used for instance to model inexpensively zero-point energy effects ([Ceriotti2](#)). This is achieved specifying the *noneq* keyword followed by the name of the file that contains the static covariance matrix for the non-equilibrium dynamics. Please note, that the covariance matrix is expected to be given in **temperature units**.

Since integrating GLE dynamics can be costly when used together with simple potentials, one can use the *every* optional keyword to apply the Langevin terms only once every several MD steps, in a multiple time-step fashion. This should be used with care when doing non-equilibrium sampling, but should have no effect on equilibrium averages when using canonical sampling.

The random number *seed* must be a positive integer. A Marsaglia random number generator is used. Each processor uses the input seed to generate its own unique seed and its own stream of random numbers. Thus the dynamics of the system will not be identical on two runs on different numbers of processors.

Note also that the Generalized Langevin Dynamics scheme that is implemented by the [fix gld](#) scheme is closely related to the present one. In fact, it should be always possible to cast the Prony series form of the memory kernel used by

GLD into an appropriate input matrix for [fix gle](#). While the GLE scheme is more general, the form used by [fix gld](#) can be more directly related to the representation of an implicit solvent environment.

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

The instantaneous values of the extended variables are written to [binary restart files](#). Because the state of the random number generator is not saved in restart files, this means you cannot do “exact” restarts with this fix, where the simulation continues on the same as if no restart had taken place. However, in a statistical sense, a restarted simulation should produce the same behavior. Note however that you should use a different seed each time you restart, otherwise the same sequence of random numbers will be used each time, which might lead to stochastic synchronization and subtle artifacts in the sampling.

The cumulative energy change in the system imposed by this fix is included in the [thermodynamic output](#) keywords `ecouple` and `econserve`. See the [thermo_style](#) doc page for details.

This fix computes a global scalar which can be accessed by various [output commands](#). The scalar is the same cumulative energy change due to this fix described in the previous paragraph. The scalar value calculated by this fix is “extensive”.

This fix can ramp its target temperature over multiple runs, using the `start` and `stop` keywords of the `run` command. See the `run` command for details of how to do this.

This fix is not invoked during [energy minimization](#).

2.72.5 Restrictions

The GLE thermostat in its current implementation should not be used with rigid bodies, SHAKE or RATTLE. It is expected that all the thermostatted degrees of freedom are fully flexible, and the sampled ensemble will not be correct otherwise.

In order to perform constant-pressure simulations please use [fix press/berendsen](#), rather than [fix npt](#), to avoid duplicate integration of the equations of motion.

This fix is part of the EXTRA-FIX package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

2.72.6 Related commands

[fix nvt](#), [fix temp/rescale](#), [fix viscous](#), [fix nvt](#), [pair_style dpd/tstat](#), [fix gld](#)

(Ceriotti) Ceriotti, Bussi and Parrinello, J Chem Theory Comput 6, 1170-80 (2010)

(GLE4MD) <https://gle4md.org/>

(Ceriotti2) Ceriotti, Bussi and Parrinello, Phys Rev Lett 103, 030603 (2009)

2.73 fix gravity command

Accelerator Variants: *gravity/omp*, *gravity/kk*

2.73.1 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)

2.73.2 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
```

2.73.3 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; its 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 ensure that the variable calculates a result in the appropriate 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.

Styles with a *gpu*, *intel*, *kk*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed on the [Accelerator packages](#) page. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the GPU, INTEL, KOKKOS, OPENMP, and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the [Build package](#) page for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the *-suffix command-line switch* when you invoke LAMMPS, or you can use the *suffix* command in your input script.

See the [Accelerator packages](#) page for more instructions on how to use the accelerated styles effectively.

2.73.4 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 global potential energy of the system as part of *thermodynamic output*. The default setting for this fix is *fix_modify energy no*.

The *fix_modify respa* option is supported by this fix. This allows to set at which level of the *r-RESPA* integrator the fix is adding its forces. Default is the outermost level.

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*.

2.73.5 Restrictions

none

2.73.6 Related commands

atom_style sphere, fix addforce

2.73.7 Default

none

2.74 fix grem command

2.74.1 Syntax

```
fix ID group-ID grem lambda eta H0 thermostat-ID
```

- ID, group-ID are documented in [fix](#) command
- grem = style name of this fix command
- lambda = intercept parameter of linear effective temperature function
- eta = slope parameter of linear effective temperature function
- H0 = shift parameter of linear effective temperature function
- thermostat-ID = ID of Nose-Hoover thermostat or barostat used in simulation

2.74.2 Examples

```
fix      ffgREM all grem 400 -0.01 -30000 fxnpt
thermo_modify  press ffgREM_press

fix      ffgREM all grem 502 -0.15 -80000 fxnvt
```

2.74.3 Description

This fix implements the molecular dynamics version of the generalized replica exchange method (gREM) originally developed by ([Kim](#)), which uses non-Boltzmann ensembles to sample over first order phase transitions. This is done by defining replicas with an enthalpy dependent effective temperature

$$T_{eff} = \lambda + \eta(H - H_0)$$

with η negative and steep enough to only intersect the characteristic microcanonical temperature (T_s) of the system once, ensuring a unimodal enthalpy distribution in that replica. λ is the intercept and effects the generalized ensemble similar to how temperature effects a Boltzmann ensemble. H_0 is a reference enthalpy, and is typically set as the lowest desired sampled enthalpy. Further explanation can be found in our recent papers ([Malolepsza](#)).

This fix requires a Nose-Hoover thermostat fix reference passed to the grem as *thermostat-ID*. Two distinct temperatures exist in this generalized ensemble, the effective temperature defined above, and a kinetic temperature that controls the velocity distribution of particles as usual. Either constant volume or constant pressure algorithms can be used.

The fix enforces a generalized ensemble in a single replica only. Typically, this ideology is combined with replica exchange with replicas differing by λ only for simplicity, but this is not required. A multi-replica simulation can

be run within the LAMMPS environment using the [temper/grem](#) command. This utilizes LAMMPS partition mode and requires the number of available processors be on the order of the number of desired replicas. A 100-replica simulation would require at least 100 processors (1 per world at minimum). If many replicas are needed on a small number of processors, multi-replica runs can be run outside of LAMMPS. An example of this can be found in examples/PACKAGES/grem and has no limit on the number of replicas per processor. However, this is very inefficient and error prone and should be avoided if possible.

In general, defining the generalized ensembles is unique for every system. When starting a many-replica simulation without any knowledge of the underlying microcanonical temperature, there are several tricks we have utilized to optimize the process. Choosing a less-steep η yields broader distributions, requiring fewer replicas to map the microcanonical temperature. While this likely struggles from the same sampling problems gREM was built to avoid, it provides quick insight to Ts. Initially using an evenly-spaced λ distribution identifies regions where small changes in enthalpy lead to large temperature changes. Replicas are easily added where needed.

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

No information about this fix is written to [binary restart files](#).

The [thermo_modify](#) press option is supported by this fix to add the rescaled kinetic pressure as part of [thermodynamic output](#).

This fix computes a global scalar which can be accessed by various [output commands](#). The scalar is the effective temperature T_{eff} . The scalar value calculated by this fix is “intensive”.

2.74.5 Restrictions

This fix is part of the REPLICA package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

2.74.6 Related commands

[temper/grem](#), [fix nvt](#), [fix npt](#), [thermo_modify](#)

2.74.7 Default

none

(Kim) Kim, Keyes, Straub, J Chem. Phys, 132, 224107 (2010).

(Malolepsza) Malolepsza, Secor, Keyes, J Phys Chem B 119 (42), 13379-13384 (2015).

2.75 fix halt command

2.75.1 Syntax

```
fix ID group-ID halt N attribute operator avalue keyword value ...
```

- ID, group-ID are documented in [fix](#) command
- halt = style name of this fix command
- N = check halt condition every N steps
- attribute = *bondmax* or *tlimit* or *v_name*

bondmax = length of longest bond in the system (in length units)
tlimit = elapsed CPU time (in seconds)
diskfree = free disk space (in MBytes)
v_name = name of [equal-style](#) variable

- operator = “<” or “<=” or “>” or “>=” or “==” or “!=” or “|^”
- avalue = numeric value to compare attribute to
- zero or more keyword/value pairs may be appended
- keyword = *error* or *message* or *path*

error value = hard or soft or continue

message value = yes or no

path value = path to check for free space (may be in quotes)

2.75.2 Examples

```
fix 10 all halt 1 bondmax > 1.5
fix 10 all halt 10 v_myCheck != 0 error soft
fix 10 all halt 100 diskfree < 100000.0 path "dump storage/."
```

2.75.3 Description

Check a condition every N steps during a simulation run. N must be ≥ 1 . If the condition is met, exit the run. In this context a “run” can be dynamics or minimization iterations, as specified by the [run](#) or [minimize](#) command.

The specified group-ID is ignored by this fix.

The specified *attribute* can be one of the options listed above, namely *bondmax*, *tlimit*, *diskfree*, or an [equal-style variable](#) referenced as *v_name*, where “name” is the name of a variable that has been defined previously in the input script.

The *bondmax* attribute will loop over all bonds in the system, compute their current lengths, and set *attribute* to the longest bond distance.

The *tlimit* attribute queries the elapsed CPU time (in seconds) since the current run began, and sets *attribute* to that value. This is an alternative way to limit the length of a simulation run, similar to the [timer](#) timeout command. There are two differences in using this method versus the timer command option. The first is that the clock starts at the beginning of the current run (not when the timer or fix command is specified), so that any setup time for the run is not included in the elapsed time. The second is that the timer invocation and syncing across all processors (via MPI_Allreduce) is not performed once every *N* steps by this command. Instead it is performed (typically) only a small number of

times and the elapsed times are used to predict when the end-of-the-run will be. Both of these attributes can be useful when performing benchmark calculations for a desired length of time with minimal overhead. For example, if a run is performing 1000s of timesteps/sec, the overhead for syncing the timer frequently across a large number of processors may be non-negligible.

The *diskfree* attribute will check for available disk space (in MBytes) on supported operating systems. By default it will check the file system of the current working directory. This can be changed with the optional *path* keyword, which will take the path to a file or folder on the file system to be checked as argument. This path must be given with single or double quotes, if it contains blanks or other special characters (like \$).

Equal-style variables evaluate to a numeric value. See the *variable* command for a description. They calculate formulas which can involve mathematical operations, atom properties, group properties, thermodynamic properties, global values calculated by a *compute* or *fix*, or references to other *variables*. Thus they are a very general means of computing some attribute of the current system. For example, the following “bondmax” variable will calculate the same quantity as the *hstyle = bondmax* option.

```
compute      bdist all bond/local dist
compute      bmax all reduce max c_bdist
variable     bondmax equal c_bmax
```

Thus these two versions of a fix halt command will do the same thing:

```
fix 10 all halt 1 bondmax > 1.5
fix 10 all halt 1 v_bondmax > 1.5
```

The version with “bondmax” will just run somewhat faster, due to less overhead in computing bond lengths and not storing them in a separate compute.

A variable can be used to implement a large variety of conditions, including to stop when a specific file exists. Example:

```
variable exit equal is_file(EXIT)
fix 10 all halt 100 v_exit != 0 error soft
```

Will stop the current run command when a file EXIT is created in the current working directory. The condition can be cleared by removing the file through the *shell* command.

The choice of operators listed above are the usual comparison operators. The XOR operation (exclusive or) is also included as “ \wedge ”. In this context, XOR means that if either the attribute or avalue is 0.0 and the other is non-zero, then the result is “true”. Otherwise it is “false”.

The specified *avalue* must be a numeric value.

The optional *error* keyword determines how the current run is halted. If its value is *hard*, then LAMMPS will stop with an error message.

If its value is *soft*, LAMMPS will exit the current run, but continue to execute subsequent commands in the input script. However, additional *run* or *minimize* commands will be skipped. For example, this allows a script to output the current state of the system, e.g. via a *write_dump* or *write_restart* command.

If its value is *continue*, the behavior is the same as for *soft*, except subsequent *run* or *minimize* commands are executed. This allows your script to remedy the condition that triggered the halt, if necessary. Note that you may wish use the *unfix* command on the fix halt ID, so that the same condition is not immediately triggered in a subsequent run.

The optional *message* keyword determines whether a message is printed to the screen and logfile when the halt condition is triggered. If *message* is set to yes, a one line message with the values that triggered the halt is printed. If *message* is set to no, no message is printed; the run simply exits. The latter may be desirable for post-processing tools that extract thermodynamic information from log files.

2.75.4 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.

2.75.5 Restrictions

The *diskfree* attribute is currently only supported on Linux, macOS, and *BSD.

2.75.6 Related commands

variable

2.75.7 Default

The option defaults are error = soft, message = yes, and path = “.”.

2.76 fix heat command

2.76.1 Syntax

```
fix ID group-ID heat N eflux
```

- ID, group-ID are documented in *fix* command
- heat = style name of this fix command
- N = add/subtract heat every this many timesteps
- eflux = rate of heat addition or subtraction (energy/time units)
- eflux 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

2.76.2 Examples

```
fix 3 qin heat 1 1.0
fix 3 qin heat 10 v_flux
fix 4 qout heat 1 -1.0 region top
```

2.76.3 Description

Add non-translational kinetic energy (heat) to a group of atoms in a manner that conserves their aggregate momentum. Two of these fixes can be used to establish a temperature gradient across a simulation domain by adding heat (energy) to one group of atoms (hot reservoir) and subtracting heat from another (cold reservoir). E.g. a simulation sampling from the McDLT ensemble.

If the *region* keyword is used, the atom must be in both the group and the specified geometric *region* in order to have energy added or subtracted to it. If not specified, then the atoms in the group are affected wherever they may move to.

Heat addition/subtraction is performed every N timesteps.

The *eflux* parameter can be specified as a numeric constant or as an equal- 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 each timestep, and its current value(s) used to determine the flux.

If *eflux* is a numeric constant or equal-style variable which evaluates to a scalar value, then *eflux* determines the change in aggregate energy of the entire group of atoms per unit time, e.g. in eV/ps for *metal units*. In this case it is an “extensive” quantity, meaning its magnitude should be scaled with the number of atoms in the group. Note that since *eflux* also has per-time units (i.e. it is a flux), this means that a larger value of N will add/subtract a larger amount of energy each time the fix is invoked.

Note

The heat-exchange (HEX) algorithm implemented by this fix is known to exhibit a pronounced energy drift. An improved algorithm (eHEX) is available as a *fix ehex* command and might be preferable if energy conservation is important.

If *eflux* is specified as an atom-style variable (see below), then the variable computes one value per atom. In this case, each value is the energy flux for a single atom, again in units of energy per unit time. In this case, each value is an “intensive” quantity, which need not be scaled with the number of atoms in the group.

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 flux.

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 flux with optional time-dependence as well.

Note

If heat is subtracted from the system too aggressively so that the group’s kinetic energy would go to zero, or any individual atom’s kinetic energy would go to zero for the case where *eflux* is an atom-style variable, then LAMMPS will halt with an error message.

Fix heat is different from a thermostat such as *fix nvt* or *fix temp/rescale* in that energy is added/subtracted continually. Thus if there is not another mechanism in place to counterbalance this effect, the entire system will heat or cool continuously. You can use multiple heat fixes so that the net energy change is 0.0 or use *fix viscous* to drain energy from the system.

This fix does not change the coordinates of its atoms; it only scales their velocities. Thus you must still use an integration fix (e.g. *fix nve*) on the affected atoms. This fix should not normally be used on atoms that have their temperature controlled by another fix - e.g. *fix nvt* or *fix langevin* fix.

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

No information about this fix is written to *binary restart files*. None of the *fix_modify* options are relevant to this fix.

This fix computes a global scalar which can be accessed by various *output commands*. This scalar is the most recent value by which velocities were scaled. The scalar value calculated by this fix is “intensive”. If *eflux* is specified as an atom-style variable, this fix computes the average value by which the velocities were scaled for all of the atoms that had their velocities scaled.

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*.

2.76.5 Restrictions

none

2.76.6 Related commands

fix ehex, compute temp, compute temp/region

2.76.7 Default

none

2.77 fix heat/flow command

2.77.1 Syntax

`fix ID group-ID heat/flow style values ...`

- ID, group-ID are documented in *fix* command
 - heat/flow = style name of this fix command
 - one style with corresponding value(s) needs to be listed
- style = constant or type
 constant = cp
 cp = value of specific heat (energy/(mass * temperature) units)
 type = cp1 ... cpN
 cpN = value of specific heat for type N (energy/(mass * temperature) units)
-

2.77.2 Examples

```
fix 1 all heat/flow constant 1.0
fix 1 all heat/flow type 1.0 0.5
```

2.77.3 Description

Perform plain time integration to update temperature for atoms in the group each timestep. The specific heat of atoms can be defined using either the *constant* or *type* keywords. For style *constant*, the specific heat is a constant value *cp* for all atoms. For style *type*, *N* different values of the specific heat are defined, one for each of the *N* types of atoms.

2.77.4 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*.

2.77.5 Restrictions

This pair style is part of the GRANULAR package. It is only enabled if LAMMPS was built with that package. See the *Build package* page for more info.

This fix requires that atoms store temperature and heat flow as defined by the *fix property/atom* command.

2.77.6 Related commands

pair granular, fix add/heat, fix property/atom

2.77.7 Default

none

2.78 fix hyper/global command

2.78.1 Syntax

```
fix ID group-ID hyper/global cutbond qfactor Vmax Tequil
```

- ID, group-ID are documented in *fix* command
- hyper/global = style name of this fix command
- cutbond = max distance at which a pair of atoms is considered bonded (distance units)
- qfactor = max strain at which bias potential goes to 0.0 (unitless)
- Vmax = height of bias potential (energy units)
- Tequil = equilibration temperature (temperature units)

2.78.2 Examples

```
fix 1 all hyper/global 1.0 0.3 0.8 300.0
```

2.78.3 Description

This fix is meant to be used with the [hyper](#) command to perform a bond-boost global hyperdynamics (GHD) simulation. The role of this fix is to select a single pair of atoms in the system at each timestep to add a global bias potential to, which will alter the dynamics of the system in a manner that effectively accelerates time. This is in contrast to the [fix hyper/local](#) command, which can be used to perform a local hyperdynamics (LHD) simulation, by adding a local bias potential to multiple pairs of atoms at each timestep. GHD can time accelerate a small simulation with up to a few 100 atoms. For larger systems, LHD is needed to achieve good time acceleration.

For a system that undergoes rare transition events, where one or more atoms move over an energy barrier to a new potential energy basin, the effect of the bias potential is to induce more rapid transitions. This can lead to a dramatic speed-up in the rate at which events occurs, without altering their relative frequencies, thus leading to an overall increase in the elapsed real time of the simulation as compared to running for the same number of timesteps with normal MD. See the [hyper](#) page for a more general discussion of hyperdynamics and citations that explain both GHD and LHD.

The equations and logic used by this fix and described here to perform GHD follow the description given in ([Voter2013](#)). The bond-boost form of a bias potential for HD is due to Miron and Fichthorn as described in ([Miron](#)). In LAMMPS we use a simplified version of bond-boost GHD where a single bond in the system is biased at any one timestep.

Bonds are defined between each pair of atoms ij , whose R_{ij}^0 distance is less than *cutbond*, when the system is in a quenched state (minimum) energy. Note that these are not “bonds” in a covalent sense. A bond is simply any pair of atoms that meet the distance criterion. *Cutbond* is an argument to this fix; it is discussed below. A bond is only formed if one or both of the ij atoms are in the specified group.

The current strain of bond ij (when running dynamics) is defined as

$$E_{ij} = \frac{R_{ij} - R_{ij}^0}{R_{ij}^0}$$

where R_{ij} is the current distance between atoms i and j , and R_{ij}^0 is the equilibrium distance in the quenched state.

The bias energy V_{ij} of any bond between atoms i and j is defined as

$$V_{ij} = V^{max} \cdot \left(1 - \left(\frac{E_{ij}}{q} \right)^2 \right) \text{ for } |E_{ij}| < qfactor \text{ or } 0 \text{ otherwise}$$

where the prefactor V^{max} and the cutoff *qfactor* are arguments to this fix; they are discussed below. This functional form is an inverse parabola centered at 0.0 with height V^{max} and which goes to 0.0 at +/- *qfactor*.

Let E^{max} be the maximum of $|E_{ij}|$ for all ij bonds in the system on a given timestep. On that step, V_{ij} is added as a bias potential to only the single bond with strain E^{max} , call it V_{ij}^{max} . Note that V_{ij}^{max} will be 0.0 if $E^{max} \geq qfactor$ on that timestep. Also note that V_{ij}^{max} is added to the normal interatomic potential that is computed between all atoms in the system at every step.

The derivative of V_{ij}^{max} with respect to the position of each atom in the E^{max} bond gives a bias force F_{ij}^{max} acting on the bond as

$$F_{ij}^{max} = -\frac{dV_{ij}^{max}}{dE_{ij}} = \frac{2V^{max}E - ij}{qfactor^2} \text{ for } |E_{ij}| < qfactor \text{ or } 0 \text{ otherwise}$$

which can be decomposed into an equal and opposite force acting on only the two ij atoms in the E^{max} bond.

The time boost factor for the system is given each timestep I by

$$B_i = e^{\beta V_{ij}^{max}}$$

where $\beta = \frac{1}{kT_{equil}}$, and T_{equil} is the temperature of the system and an argument to this fix. Note that $B_i \geq 1$ at every step.

Note

To run a GHD simulation, the input script must also use the [fix langevin](#) command to thermostat the atoms at the same T_{equil} as specified by this fix, so that the system is running constant-temperature (NVT) dynamics. LAMMPS does not check that this is done.

The elapsed time t_{hyper} for a GHD simulation running for N timesteps is simply

$$t_{hyper} = \sum_{i=1,N} B - i \cdot dt$$

where dt is the timestep size defined by the [timestep](#) command. The effective time acceleration due to GHD is thus $t_{hyper} / N * dt$, where $N * dt$ is elapsed time for a normal MD run of N timesteps.

Note that in GHD, the boost factor varies from timestep to timestep. Likewise, which bond has E^{max} strain and thus which pair of atoms the bias potential is added to, will also vary from timestep to timestep. This is in contrast to local hyperdynamics (LHD) where the boost factor is an input parameter; see the [fix hyper/local](#) page for details.

Here is additional information on the input parameters for GHD.

The *cutbond* argument is the cutoff distance for defining bonds between pairs of nearby atoms. A pair of *ij* atoms in their equilibrium, minimum-energy configuration, which are separated by a distance $R_{ij} < cutbond$, are flagged as a bonded pair. Setting *cutbond* to be ~25% larger than the nearest-neighbor distance in a crystalline lattice is a typical choice for solids, so that bonds exist only between nearest neighbor pairs.

The *qfactor* argument is the limiting strain at which the bias potential goes to 0.0. It is dimensionless, so a value of 0.3 means a bond distance can be up to 30% larger or 30% smaller than the equilibrium (quenched) R_{ij}^0 distance and the two atoms in the bond could still experience a non-zero bias force.

If *qfactor* is set too large, then transitions from one energy basin to another are affected because the bias potential is non-zero at the transition state (e.g. saddle point). If *qfactor* is set too small than little boost is achieved because the E_{ij} strain of some bond in the system will (nearly) always exceed *qfactor*. A value of 0.3 for *qfactor* is typically reasonable.

The *Vmax* argument is the prefactor on the bias potential. Ideally, it should be set to a value slightly less than the smallest barrier height for an event to occur. Otherwise the applied bias potential may be large enough (when added to the interatomic potential) to produce a local energy basin with a maxima in the center. This can produce artificial energy minima in the same basin that trap an atom. Or if *Vmax* is even larger, it may induce an atom(s) to rapidly transition to another energy basin. Both cases are “bad dynamics” which violate the assumptions of GHD that guarantee an accelerated time-accurate trajectory of the system.

Note that if *Vmax* is set too small, the GHD simulation will run correctly. There will just be fewer events because the hyper time (t_{hyper} equation above) will be shorter.

Note

If you have no physical intuition as to the smallest barrier height in your system, a reasonable strategy to determine the largest *Vmax* you can use for a GHD model, is to run a sequence of simulations with smaller and smaller *Vmax* values, until the event rate does not change (as a function of hyper time).

The *Tequil* argument is the temperature at which the system is simulated; see the comment above about the [fix langevin](#) thermostatting. It is also part of the beta term in the exponential factor that determines how much boost is achieved as a function of the bias potential.

In general, the lower the value of *Tequil* and the higher the value of *Vmax*, the more time boost will be achievable by the GHD algorithm.

2.78.4 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 the bias potential to the global potential energy of the system as part of [thermodynamic output](#). The default setting for this fix is *fix_modify energy no*.

This fix computes a global scalar and global vector of length 12, which can be accessed by various [output commands](#). The scalar is the magnitude of the bias potential (energy units) applied on the current timestep. The vector stores the following quantities:

1. boost factor on this step (unitless)
2. max strain E_{ij} of any bond on this step (absolute value, unitless)
3. ID of first atom in the max-strain bond
4. ID of second atom in the max-strain bond
5. average # of bonds/atom on this step
6. fraction of timesteps where the biased bond has bias = 0.0 during this run
7. fraction of timesteps where the biased bond has negative strain during this run
8. max drift distance of any atom during this run (distance units)
9. max bond length during this run (distance units)
10. cumulative hyper time since fix was defined (time units)
11. cumulative count of event timesteps since fix was defined
12. cumulative count of atoms in events since fix was defined

The first 5 quantities are for the current timestep. Quantities 6-9 are for the current hyper run. They are reset each time a new hyper run is performed. Quantities 10-12 are cumulative across multiple runs (since the point in the input script the fix was defined).

For value 8, drift is the distance an atom moves between two quenched states when the second quench determines an event has occurred. Atoms involved in an event will typically move the greatest distance since others typically remain near their original quenched position.

For value 11, events are checked for by the [hyper](#) command once every *Nevent* timesteps. This value is the count of those timesteps on which one (or more) events was detected. It is NOT the number of distinct events, since more than one event may occur in the same *Nevent* time window.

For value 12, each time the [hyper](#) command checks for an event, it invokes a compute to flag zero or more atoms as participating in one or more events. E.g. atoms that have displaced more than some distance from the previous quench state. Value 11 is the cumulative count of the number of atoms participating in any of the events that were found.

The scalar and vector values calculated by this fix are all “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](#).

2.78.5 Restrictions

This command can only be used if LAMMPS was built with the REPLICA package. See the [Build package](#) page for more info.

2.78.6 Related commands

hyper, fix hyper/local

2.78.7 Default

none

(**Voter2013**) S. Y. Kim, D. Perez, A. F. Voter, J Chem Phys, 139, 144110 (2013).

(**Miron**) R. A. Miron and K. A. Fichthorn, J Chem Phys, 119, 6210 (2003).

2.79 fix hyper/local command

2.79.1 Syntax

```
fix ID group-ID hyper/local cutbond qfactor Vmax Tequil Dcut alpha Btarget
```

- ID, group-ID are documented in [fix](#) command
- hyper/local = style name of this fix command
- cutbond = max distance at which a pair of atoms is considered bonded (distance units)
- qfactor = max strain at which bias potential goes to 0.0 (unitless)
- Vmax = estimated height of bias potential (energy units)
- Tequil = equilibration temperature (temperature units)
- Dcut = minimum distance between boosted bonds (distance units)
- alpha = boostostat relaxation time (time units)
- Btarget = desired time boost factor (unitless)
- zero or more keyword/value pairs may be appended
- keyword = *bound* or *reset* or *check/ghost* or *check/bias*

bound value = Bfrac

Bfrac = -1 or a value ≥ 0.0

reset value = Rfreq

Rfreq = -1 or 0 or timestep value > 0

check/ghost values = none

check/bias values = Nevery error/warn/ignore

2.79.2 Examples

```
fix 1 all hyper/local 1.0 0.3 0.8 300.0
fix 1 all hyper/local 1.0 0.3 0.8 300.0 bound 0.1 reset 0
```

2.79.3 Description

This fix is meant to be used with the [hyper](#) command to perform a bond-boost local hyperdynamics (LHD) simulation. The role of this fix is to select multiple pairs of atoms in the system at each timestep to add a local bias potential to, which will alter the dynamics of the system in a manner that effectively accelerates time. This is in contrast to the [fix hyper/global](#) command, which can be used to perform a global hyperdynamics (GHD) simulation, by adding a global bias potential to a single pair of atoms at each timestep. GHD can time accelerate a small simulation with up to a few 100 atoms. For larger systems, LHD is needed to achieve good time acceleration.

For a system that undergoes rare transition events, where one or more atoms move over an energy barrier to a new potential energy basin, the effect of the bias potential is to induce more rapid transitions. This can lead to a dramatic speed-up in the rate at which events occurs, without altering their relative frequencies, thus leading to an overall increase in the elapsed real time of the simulation as compared to running for the same number of timesteps with normal MD. See the [hyper](#) page for a more general discussion of hyperdynamics and citations that explain both GHD and LHD.

The equations and logic used by this fix and described here to perform LHD follow the description given in ([Voter2013](#)). The bond-boost form of a bias potential for HD is due to Miron and Fichthorn as described in ([Miron](#)).

To understand this description, you should first read the description of the GHD algorithm on the [fix hyper/global](#) doc page. This description of LHD builds on the GHD description.

The definition of bonds and E_{ij} are the same for GHD and LHD. The formulas for V_{ij}^{max} and F_{ij}^{max} are also the same except for a prefactor C_{ij} , explained below.

The bias energy V_{ij} applied to a bond ij with maximum strain is

$$V_{ij}^{max} = C_{ij} \cdot V^{max} \cdot \left(1 - \left(\frac{E_{ij}}{q} \right)^2 \right) \text{ for } |E_{ij}| < q\text{factor} \text{ or } 0 \text{ otherwise}$$

The derivative of V_{ij}^{max} with respect to the position of each atom in the ij bond gives a bias force F_{ij}^{max} acting on the bond as

$$F_{ij}^{max} = -\frac{dV_{ij}^{max}}{dE_{ij}} = 2C_{ij}V^{max} \frac{E_{ij}}{q\text{factor}^2} \text{ for } |E_{ij}| < q\text{factor} \text{ or } 0 \text{ otherwise}$$

which can be decomposed into an equal and opposite force acting on only the two atoms i and j in the ij bond.

The key difference is that in GHD a bias energy and force is added (on a particular timestep) to only one bond (pair of atoms) in the system, which is the bond with maximum strain E^{max} .

In LHD, a bias energy and force can be added to multiple bonds separated by the specified $Dcut$ distance or more. A bond ij is biased if it is the maximum strain bond within its local “neighborhood”, which is defined as the bond ij plus any neighbor bonds within a distance $Dcut$ from ij . The “distance” between bond ij and bond kl is the minimum distance between any of the ik , il , jk , and jl pairs of atoms.

For a large system, multiple bonds will typically meet this requirement, and thus a bias potential V_{ij}^{max} will be applied to many bonds on the same timestep.

In LHD, all bonds store a C_{ij} prefactor which appears in the V_{ij}^{max} and F_{ij}^{max} equations above. Note that the C_{ij} factor scales the strength of the bias energy and forces whenever bond ij is the maximum strain bond in its neighborhood.

C_{ij} is initialized to 1.0 when a bond between the ij atoms is first defined. The specified $Btarget$ factor is then used to adjust the C_{ij} prefactors for each bond every timestep in the following manner.

An instantaneous boost factor B_{ij} is computed each timestep for each bond, as

$$B_{ij} = e^{\beta V_{kl}^{max}}$$

where V_{kl}^{max} is the bias energy of the maxstrain bond kl within bond ij 's neighborhood, $\beta = \frac{1}{kT_{equil}}$, and T_{equil} is the temperature of the system and an argument to this fix.

Note

To run an LHD simulation, the input script must also use the [fix langevin](#) command to thermostat the atoms at the same T_{equil} as specified by this fix, so that the system is running constant-temperature (NVT) dynamics. LAMMPS does not check that this is done.

Note that if $ij == kl$, then bond ij is a biased bond on that timestep, otherwise it is not. But regardless, the boost factor B_{ij} can be thought of an estimate of time boost currently being applied within a local region centered on bond ij . For LHD, we want this to be the specified B_{target} value everywhere in the simulation domain.

To accomplish this, if $B_{ij} < B_{target}$, the C_{ij} prefactor for bond ij is incremented on the current timestep by an amount proportional to the inverse of the specified α and the difference $(B_{ij} - B_{target})$. Conversely if $B_{ij} > B_{target}$, C_{ij} is decremented by the same amount. This procedure is termed “boostostatting” in ([Voter2013](#)). It drives all of the individual C_{ij} to values such that when V_{ij}^{max} is applied as a bias to bond ij , the resulting boost factor B_{ij} will be close to B_{target} on average. Thus the LHD time acceleration factor for the overall system is effectively B_{target} .

Note that in LHD, the boost factor B_{target} is specified by the user. This is in contrast to global hyperdynamics (GHD) where the boost factor varies each timestep and is computed as a function of V_{max} , E_{max} , and T_{equil} ; see the [fix hyper/global](#) page for details.

Here is additional information on the input parameters for LHD.

Note that the *cutbond*, *qfactor*, and *Tequil* arguments have the same meaning as for GHD. The *Vmax* argument is slightly different. The *Dcut*, *alpha*, and *Btarget* parameters are unique to LHD.

The *cutbond* argument is the cutoff distance for defining bonds between pairs of nearby atoms. A pair of I,J atoms in their equilibrium, minimum-energy configuration, which are separated by a distance $R_{ij} < cutbond$, are flagged as a bonded pair. Setting *cutbond* to be ~25% larger than the nearest-neighbor distance in a crystalline lattice is a typical choice for solids, so that bonds exist only between nearest neighbor pairs.

The *qfactor* argument is the limiting strain at which the bias potential goes to 0.0. It is dimensionless, so a value of 0.3 means a bond distance can be up to 30% larger or 30% smaller than the equilibrium (quenched) R_{ij}^0 distance and the two atoms in the bond could still experience a non-zero bias force.

If *qfactor* is set too large, then transitions from one energy basin to another are affected because the bias potential is non-zero at the transition state (e.g. saddle point). If *qfactor* is set too small than little boost can be achieved because the E_{ij} strain of some bond in the system will (nearly) always exceed *qfactor*. A value of 0.3 for *qfactor* is typically a reasonable value.

The *Vmax* argument is a fixed prefactor on the bias potential. There is also a dynamic prefactor C_{ij} , driven by the choice of *Btarget* as discussed above. The product of these should be a value less than the smallest barrier height for an event to occur. Otherwise the applied bias potential may be large enough (when added to the interatomic potential) to produce a local energy basin with a maxima in the center. This can produce artificial energy minima in the same basin that trap an atom. Or if $C_{ij} \cdot V^{max}$ is even larger, it may induce an atom(s) to rapidly transition to another energy basin. Both cases are “bad dynamics” which violate the assumptions of LHD that guarantee an accelerated time-accurate trajectory of the system.

Note

It may seem that V^{max} can be set to any value, and C_{ij} will compensate to reduce the overall prefactor if necessary. However the C_{ij} are initialized to 1.0 and the booststatting procedure typically operates slowly enough that there can be a time period of bad dynamics if V^{max} is set too large. A better strategy is to set V^{max} to the slightly smaller than the lowest barrier height for an event (the same as for GHD), so that the C_{ij} remain near unity.

The *Tequil* argument is the temperature at which the system is simulated; see the comment above about the [fix langevin](#) thermostatting. It is also part of the beta term in the exponential factor that determines how much boost is achieved as a function of the bias potential. See the discussion of the *Btarget* argument below.

As discussed above, the *Dcut* argument is the distance required between two locally maxstrain bonds for them to both be selected as biased bonds on the same timestep. Computationally, the larger *Dcut* is, the more work (computation and communication) must be done each timestep within the LHD algorithm. And the fewer bonds can be simultaneously biased, which may mean the specified *Btarget* time acceleration cannot be achieved.

Physically *Dcut* should be a long enough distance that biasing two pairs of atoms that close together will not influence the dynamics of each pair. E.g. something like 2x the cutoff of the interatomic potential. In practice a *Dcut* value of ~10 Angstroms seems to work well for many solid-state systems.

Note

You should ensure that ghost atom communication is performed for a distance of at least $Dcut + cutevent$ = the distance one or more atoms move (between quenched states) to be considered an “event”. It is an argument to the “compute event/displace” command used to detect events. By default the ghost communication distance is set by the pair_style cutoff, which will typically be $< Dcut$. The [comm_modify cutoff](#) command should be used to override the ghost cutoff explicitly, e.g.

```
comm_modify cutoff 12.0
```

Note that this fix does not know the *cutevent* parameter, but uses half the *cutbond* parameter as an estimate to warn if the ghost cutoff is not long enough.

As described above the *alpha* argument is a prefactor in the booststat update equation for each bond’s C_{ij} prefactor. *Alpha* is specified in time units, similar to other thermostat or barostat damping parameters. It is roughly the physical time it will take the booststat to adjust a C_{ij} value from a too high (or too low) value to a correct one. An *alpha* setting of a few ps is typically good for solid-state systems. Note that the *alpha* argument here is the inverse of the alpha parameter discussed in ([Voter2013](#)).

The *Btarget* argument is the desired time boost factor (a value > 1) that all the atoms in the system will experience. The elapsed time *t_hyper* for an LHD simulation running for *N* timesteps is simply

$$t_{hyper} = B_{target} \cdot N \cdot dt$$

where *dt* is the timestep size defined by the [timestep](#) command. The effective time acceleration due to LHD is thus $\frac{t_{hyper}}{N \cdot dt} = B_{target}$, where $N \cdot dt$ is the elapsed time for a normal MD run of *N* timesteps.

You cannot choose an arbitrarily large setting for *Btarget*. The maximum value you should choose is

$$B_{target} = e^{\beta V_{small}}$$

where V_{small} is the smallest event barrier height in your system, $\beta = \frac{1}{kT_{equil}}$, and T_{equil} is the specified temperature of the system (both by this fix and the Langevin thermostat).

Note that if *Btarget* is set smaller than this, the LHD simulation will run correctly. There will just be fewer events because the hyper time (*t_hyper* equation above) will be shorter.

 **Note**

If you have no physical intuition as to the smallest barrier height in your system, a reasonable strategy to determine the largest *Btarget* you can use for an LHD model, is to run a sequence of simulations with smaller and smaller *Btarget* values, until the event rate does not change (as a function of hyper time).

Here is additional information on the optional keywords for this fix.

The *bound* keyword turns on min/max bounds for bias coefficients C_{ij} for all bonds. C_{ij} is a prefactor for each bond on the bias potential of maximum strength V^{max} . Depending on the choice of *alpha* and *Btarget* and *Vmax*, the boostostatting can cause individual C_{ij} values to fluctuate. If the fluctuations are too large $C_{ij} \cdot V^{max}$ can exceed low barrier heights and induce bad event dynamics. Bounding the C_{ij} values is a way to prevent this. If *Bfrac* is set to -1 or any negative value (the default) then no bounds are enforced on C_{ij} values (except they must always be ≥ 0.0). A *Bfrac* setting ≥ 0.0 sets a lower bound of $1.0 - Bfrac$ and upper bound of $1.0 + Bfrac$ on each C_{ij} value. Note that all C_{ij} values are initialized to 1.0 when a bond is created for the first time. Thus *Bfrac* limits the bias potential height to $Vmax \pm Bfrac * Vmax$.

The *reset* keyword allow *Vmax* to be adjusted dynamically depending on the average value of all C_{ij} prefactors. This can be useful if you are unsure what value of *Vmax* will match the *Btarget* boost for the system. The C_{ij} values will then adjust in aggregate (up or down) so that $C_{ij} \cdot V^{max}$ produces a boost of *Btarget*, but this may conflict with the *bound* keyword settings. By using *bound* and *reset* together, *Vmax* itself can be reset, and desired bounds still applied to the C_{ij} values.

A setting for *Rfreq* of -1 (the default) means *Vmax* never changes. A setting of 0 means *Vmax* is adjusted every time an event occurs and bond pairs are recalculated. A setting of *N* > 0 timesteps means *Vmax* is adjusted on the first time an event occurs on a timestep $\geq N$ steps after the previous adjustment. The adjustment to *Vmax* is computed as follows. The current average of all $C_{ij} \cdot V^{max}$ values is computed and the *Vmax* is reset to that value. All C_{ij} values are changed to new prefactors such the new $C_{ij} \cdot V^{max}$ is the same as it was previously. If the *bound* keyword was used, those bounds are enforced on the new C_{ij} values. Henceforth, new bonds are assigned a $C_{ij} = 1.0$, which means their bias potential magnitude is the new *Vmax*.

The *check/ghost* keyword turns on extra computation each timestep to compute statistics about ghost atoms used to determine which bonds to bias. The output of these stats are the vector values 14 and 15, described below. If this keyword is not enabled, the output of the stats will be zero.

The *check/bias* keyword turns on extra computation and communication to check if any biased bonds are closer than *Dcut* to each other, which should not be the case if LHD is operating correctly. Thus it is a debugging check. The *Nevery* setting determines how often the check is made. The *error*, *warn*, or *ignore* setting determines what is done if the count of too-close bonds is not zero. Either the code will exit, or issue a warning, or silently tally the count. The count can be output as vector value 17, as described below. If this keyword is not enabled, the output of that statistic will be 0.

Note that both of these computations are costly, hence they are only enabled by these keywords.

2.79.4 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 the bias potential to the global potential energy of the system as part of *thermodynamic output*. The default setting for this fix is *fix_modify energy no*.

This fix computes a global scalar and global vector of length 28, which can be accessed by various *output commands*. The scalar is the magnitude of the bias potential (energy units) applied on the current timestep, summed over all biased bonds. The vector stores the following quantities:

1. average boost for all bonds on this step (unitless)
2. # of biased bonds on this step
3. max strain E_{ij} of any bond on this step (absolute value, unitless)
4. value of V^{max} on this step (energy units)
5. average bias coeff for all bonds on this step (unitless)
6. min bias coeff for all bonds on this step (unitless)
7. max bias coeff for all bonds on this step (unitless)
8. average # of bonds/atom on this step
9. average neighbor bonds/bond on this step within D_{cut}
10. average boost for all bonds during this run (unitless)
11. average # of biased bonds/step during this run
12. fraction of biased bonds with no bias during this run
13. fraction of biased bonds with negative strain during this run
14. max bond length during this run (distance units)
15. average bias coeff for all bonds during this run (unitless)
16. min bias coeff for any bond during this run (unitless)
17. max bias coeff for any bond during this run (unitless)
18. max drift distance of any bond atom during this run (distance units)
19. max distance from proc subbox of any ghost atom with maxstrain < qfactor during this run (distance units)
20. max distance outside my box of any ghost atom with any maxstrain during this run (distance units)
21. count of ghost atoms that could not be found on reneighbor steps during this run
22. count of bias overlaps (< D_{cut}) found during this run
23. cumulative hyper time since fix created (time units)
24. cumulative count of event timesteps since fix created
25. cumulative count of atoms in events since fix created
26. cumulative # of new bonds formed since fix created
27. average boost for biased bonds on this step (unitless)
28. # of bonds with absolute strain $\geq q$ on this step

Quantities 1-9 are for the current timestep. Quantities 10-22 are for the current hyper run. They are reset each time a new hyper run is performed. Quantities 23-26 are cumulative across multiple runs (since the point in the input script the fix was defined).

For value 10, each bond instantaneous boost factor is given by the equation for B_{ij} above. The total system boost (average across all bonds) fluctuates, but should average to a value close to the specified B_{target} .

For value 12, the numerator is a count of all biased bonds on each timestep whose bias energy = 0.0 due to $E_{ij} \geq qfactor$. The denominator is the count of all biased bonds on all timesteps.

For value 13, the numerator is a count of all biased bonds on each timestep with negative strain. The denominator is the count of all biased bonds on all timesteps.

Values 18-22 are mostly useful for debugging and diagnostic purposes.

For value 18, drift is the distance an atom moves between two quenched states when the second quench determines an event has occurred. Atoms involved in an event will typically move the greatest distance since others typically remain near their original quenched position.

For values 19-21, neighbor atoms in the full neighbor list with cutoff $Dcut$ may be ghost atoms outside a processor's sub-box. Before the next event occurs they may move further than $Dcut$ away from the sub-box boundary. Value 19 is the furthest (from the sub-box) any ghost atom in the neighbor list with maxstrain < $qfactor$ was accessed during the run. Value 20 is the same except that the ghost atom's maxstrain may be $\geq qfactor$, which may mean it is about to participate in an event. Value 21 is a count of how many ghost atoms could not be found on renighbor steps, presumably because they moved too far away due to their participation in an event (which will likely be detected at the next quench).

Typical values for 19 and 20 should be slightly larger than $Dcut$, which accounts for ghost atoms initially at a $Dcut$ distance moving thermally before the next event takes place.

Note that for values 19 and 20 to be computed, the optional keyword *check/ghost* must be specified. Otherwise these values will be zero. This is because computing them incurs overhead, so the values are only computed if requested.

Value 21 should be zero or small. As explained above a small count likely means some ghost atoms were participating in their own events and moved a longer distance. If the value is large, it likely means the communication cutoff for ghosts is too close to $Dcut$ leading to many not-found ghost atoms before the next event. This may lead to a reduced number of bonds being selected for biasing, since the code assumes those atoms are part of highly strained bonds. As explained above, the *comm_modify cutoff* command can be used to set a longer cutoff.

For value 22, no two bonds should be biased if they are within a $Dcut$ distance of each other. This value should be zero, indicating that no pair of biased bonds are closer than $Dcut$ from each other.

Note that for value 22 to be computed, the optional keyword *check/bias* must be specified and it determines how often this check is performed. This is because performing the check incurs overhead, so if only computed as often as requested.

The result at the end of the run is the cumulative total from every timestep the check was made. Note that the value is a count of atoms in bonds which found other atoms in bonds too close, so it is almost always an over-count of the number of too-close bonds.

Value 23 is simply the specified *boost* factor times the number of timesteps times the timestep size.

For value 24, events are checked for by the *hyper* command once every $Nevent$ timesteps. This value is the count of those timesteps on which one (or more) events was detected. It is NOT the number of distinct events, since more than one event may occur in the same $Nevent$ time window.

For value 25, each time the *hyper* command checks for an event, it invokes a compute to flag zero or more atoms as participating in one or more events. E.g. atoms that have displaced more than some distance from the previous quench state. Value 25 is the cumulative count of the number of atoms participating in any of the events that were found.

Value 26 tallies the number of new bonds created by the bond reset operation. Bonds between a specific I,J pair of atoms may persist for the entire hyperdynamics simulation if neither I or J are involved in an event.

Value 27 computes the average boost for biased bonds only on this step.

Value 28 is the count of bonds with an absolute value of strain $\geq q$ on this step.

The scalar value and vector values are all “intensive”.

This fix also computes a local vector of length the number of bonds currently in the system. The value for each bond is its C_{ij} prefactor (bias coefficient). These values can be accessed by various *output commands*. A particularly useful one is the *fix ave/histo* command which can be used to histogram the C_{ij} values to see if they are distributed reasonably close to 1.0, which indicates a good choice of V^{max} .

The local values calculated by this fix are unitless.

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*.

2.79.5 Restrictions

This fix is part of the REPLIC^A package. It is only enabled if LAMMPS was built with that package. See the *Build package* doc page for more info.

2.79.6 Related commands

hyper, *fix hyper/global*

2.79.7 Default

The default settings for optimal keywords are bounds = -1 and reset = -1. The check/ghost and check/bias keywords are not enabled by default.

(Voter2013) S. Y. Kim, D. Perez, A. F. Voter, J Chem Phys, 139, 144110 (2013).

(Miron) R. A. Miron and K. A. Fichthorn, J Chem Phys, 119, 6210 (2003).

2.80 fix imd command

2.80.1 Syntax

```
fix ID group-ID imd rate port keyword values ...
```

- ID, group-ID are documented in *fix* command
- imd = style name of this fix command
- port = port number on which the fix listens for an IMD client
- keyword = *unwrap* or *fscale* or *rate* or *nowait*

unwrap arg = on or off

off = coordinates are wrapped back into the principal unit cell (default)

on = "unwrapped" coordinates using the image flags used

fscale arg = factor

factor = floating point number to scale IMD forces (default: 1.0)

rate arg = transmission rate of coordinate data sets (default: 1)
nowait arg = on or off
off = LAMMPS waits to be connected to an IMD client before continuing (default)
on = LAMMPS listens for an IMD client, but continues with the run

2.80.2 Examples

```
fix vmd all md 5678
fix comm all md 8888 rate 5 unwrap on fscale 10.0
```

2.80.3 Description

This fix implements the “Interactive MD” (IMD) protocol which allows realtime visualization and manipulation of MD simulations through the IMD protocol, as initially implemented in VMD and NAMD. Specifically it allows LAMMPS to connect an IMD client, for example the [VMD visualization program](#), so that it can monitor the progress of the simulation and interactively apply forces to selected atoms.

If LAMMPS is compiled with the pre-processor flag -DLAMMPS_ASYNC_IMD then fix md will use POSIX threads to spawn a IMD communication thread on MPI rank 0 in order to offload data reading and writing from the main execution thread and potentially lower the inferred latencies for slow communication links. This feature has only been tested under linux.

The source code for this fix includes code developed by the Theoretical and Computational Biophysics Group in the Beckman Institute for Advanced Science and Technology at the University of Illinois at Urbana-Champaign. We thank them for providing a software interface that allows codes like LAMMPS to hook to [VMD](#).

Upon initialization of the fix, it will open a communication port on the node with MPI task 0 and wait for an incoming connection. As soon as an IMD client is connected, the simulation will continue and the fix will send the current coordinates of the fix’s group to the IMD client at every rate MD step. When using r-RESPA, rate applies to the steps of the outmost RESPA level. During a run with an active IMD connection also the IMD client can request to apply forces to selected atoms of the fix group.

The port number selected must be an available network port number. On many machines, port numbers < 1024 are reserved for accounts with system manager privilege and specific applications. If multiple md fixes would be active at the same time, each needs to use a different port number.

The *nowait* keyword controls the behavior of the fix when no IMD client is connected. With the default setting of *off*, LAMMPS will wait until a connection is made before continuing with the execution. Setting *nowait* to *on* will have the LAMMPS code be ready to connect to a client, but continue with the simulation. This can for example be used to monitor the progress of an ongoing calculation without the need to be permanently connected or having to download a trajectory file.

The *rate* keyword allows to select how often the coordinate data is sent to the IMD client. It can also be changed on request of the IMD client through an IMD protocol message. The *unwrap* keyword allows to send “unwrapped” coordinates to the IMD client that undo the wrapping back of coordinates into the principle unit cell, as done by default in LAMMPS. The *fscale* keyword allows to apply a scaling factor to forces transmitted by the IMD client. The IMD protocols stipulates that forces are transferred in kcal/mol/Angstrom under the assumption that coordinates are given in Angstrom. For LAMMPS runs with different units or as a measure to tweak the forces generated by the manipulation of the IMD client, this option allows to make adjustments.

To connect VMD to a listening LAMMPS simulation on the same machine with fix md enabled, one needs to start VMD and load a coordinate or topology file that matches the fix group. When the VMD command prompt appears, one types the command line:

```
imd connect localhost 5678
```

This assumes that *fix IMD* was started with 5678 as a port number for the IMD protocol.

The steps to do interactive manipulation of a running simulation in VMD are the following:

In the Mouse menu of the VMD Main window, select “Mouse -> Force -> Atom”. You may alternately select “Residue”, or “Fragment” to apply forces to whole residues or fragments. Your mouse can now be used to apply forces to your simulation. Click on an atom, residue, or fragment and drag to apply a force. Click quickly without moving the mouse to turn the force off. You can also use a variety of 3D position trackers to apply forces to your simulation. Game controllers or haptic devices with force-feedback such as the Novint Falcon or Sensable PHANTOM allow you to feel the resistance due to inertia or interactions with neighbors that the atoms experience you are trying to move, as if they were real objects. See the [VMD IMD Homepage](#) for more details.

If IMD control messages are received, a line of text describing the message and its effect will be printed to the LAMMPS output screen, if screen output is active.

2.80.4 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*.

2.80.5 Restrictions

This fix is part of the MISC package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

When used in combination with VMD, a topology or coordinate file has to be loaded, which matches (in number and ordering of atoms) the group the fix is applied to. The fix internally sorts atom IDs by ascending integer value; in VMD (and thus the IMD protocol) those will be assigned 0-based consecutive index numbers.

When using multiple active IMD connections at the same time, each needs to use a different port number.

2.80.6 Related commands

none

2.80.7 Default

none

2.81 fix indent command

2.81.1 Syntax

```
fix ID group-ID indent K gstyle args keyword value ...
```

- ID, group-ID are documented in [fix](#) command
- indent = style name of this fix command
- K = force constant for indenter surface (force/distance² units)
- gstyle = *sphere* or *cylinder* or *cone* or *plane*

sphere args = x y z R

 x, y, z = position of center of indenter (distance units)

 R = sphere radius of indenter (distance units)

 any of x, y, z, R can be a variable (see below)

cylinder args = dim c1 c2 R

 dim = x or y or z = axis of cylinder

 c1, c2 = coords of cylinder axis in other 2 dimensions (distance units)

 R = cylinder radius of indenter (distance units)

 any of c1,c2,R can be a variable (see below)

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)

 any of c1, c2, radlo, radhi, lo, hi can be a variable (see below)

plane args = dim pos side

 dim = x or y or z = plane perpendicular to this dimension

 pos = position of plane in dimension x, y, or z (distance units)

 pos can be a variable (see below)

 side = lo or hi

- zero or more keyword/value pairs may be appended

- keyword = *side* or *units*

side value = in or out

 in = the indenter acts on particles inside the sphere or cylinder or cone

 out = the indenter acts on particles outside the sphere or cylinder or cone

units value = lattice or box

 lattice = the geometry is defined in lattice units

 box = the geometry is defined in simulation box units

2.81.2 Examples

```
fix 1 all indent 10.0 sphere 0.0 0.0 15.0 3.0
fix 1 all indent 10.0 sphere v_x v_y 0.0 v_radius side in
fix 2 flow indent 10.0 cylinder z 0.0 0.0 10.0 units box
```

2.81.3 Description

Insert an indenter within a simulation box. The indenter repels all atoms in the group that touch it, so it can be used to push into a material or as an obstacle in a flow. Alternatively, it can be used as a constraining wall around a simulation; see the discussion of the *side* keyword below.

The *gstyle* keyword selects the geometry of the indenter and it can either have the value of *sphere*, *cylinder*, *cone*, or *plane*.

A spherical indenter (*gstyle = sphere*) exerts a force of magnitude

$$F(r) = -K(r - R)^2$$

on each atom where *K* is the specified force constant, *r* is the distance from the atom to the center of the indenter, and *R* is the radius of the indenter. The force is repulsive and $F(r) = 0$ for $r > R$.

A cylindrical indenter (*gstyle = cylinder*) follows the same formula for the force as a sphere, except that *r* is defined the distance from the atom to the center axis of the cylinder. The cylinder extends infinitely along its axis.

Added in version 17April2024.

A conical indenter (*gstyle = cone*) is similar to a cylindrical indenter except that it has a finite length (between *lo* and *hi*), and that two different radii (one at each end, *radlo* and *radhi*) can be defined.

Spherical, cylindrical, and conical indenters account for periodic boundaries in two ways. First, the center point of a spherical indenter (x,y,z) or axis of a cylindrical/conical indenter (c1,c2) is remapped back into the simulation box, if the box is periodic in a particular dimension. This occurs every timestep if the indenter geometry is specified with a variable (see below), e.g. it is moving over time. Second, the calculation of distance to the indenter center or axis accounts for periodic boundaries. Both of these mean that an indenter can effectively move through and straddle one or more periodic boundaries.

A planar indenter (*gstyle = plane*) behaves like an axis-aligned infinite-extent wall with the same force expression on atoms in the system as before, but where *R* is the position of the plane and *r-R* is the distance of an atom from the plane. If the *side* parameter of the plane is specified as *lo* then it will indent from the *lo* end of the simulation box, meaning that atoms with a coordinate less than the plane's current position will be pushed towards the *hi* end of the box and atoms with a coordinate higher than the plane's current position will feel no force. Vice versa if *side* is specified as *hi*.

Any of the 4 quantities defining a spherical indenter's geometry can be specified as an equal-style *variable*, namely *x*, *y*, *z*, or *R*. For a cylindrical indenter, any of the 3 quantities *c1*, *c2*, or *R*, can be a variable. For a conical indenter, any of the 6 quantities *c1*, *c2*, *radlo*, *radhi*, *lo*, or *hi* can be a variable. For a planar indenter, the single value *pos* can be a variable.

If any of these values is a variable, it should be specified as *v_name*, where *name* is the variable name. In this case, the variable will be evaluated each timestep, and its value used to define the indenter geometry.

Note that equal-style variables can specify formulas with various mathematical functions, and include *thermo_style* command keywords for the simulation box parameters and timestep and elapsed time. Thus it is easy to specify indenter properties that change as a function of time or span consecutive runs in a continuous fashion. For the latter, see the *start* and *stop* keywords of the *run* command and the *elaplong* keyword of *thermo_style custom* for details.

For example, if a spherical indenter's x-position is specified as *v_x*, then this variable definition will keep its center at a relative position in the simulation box, 1/4 of the way from the left edge to the right edge, even if the box size changes:

```
variable x equal "xlo + 0.25*lx"
```

Similarly, either of these variable definitions will move the indenter from an initial position at 2.5 at a constant velocity of 5:

```
variable x equal "2.5 + 5*elaplong*dt"
variable x equal vdisplace(2.5,5)
```

If a spherical indenter's radius is specified as `v_r`, then these variable definitions will grow the size of the indenter at a specified rate.

```
variable r0 equal 0.0
variable rate equal 1.0
variable r equal "v_r0 + step*dt*v_rate"
```

If the `side` keyword is specified as `out`, which is the default, then particles outside the indenter are pushed away from its outer surface, as described above. This only applies to spherical, cylindrical, and conical indenters. If the `side` keyword is specified as `in`, the action of the indenter is reversed. Particles inside the indenter are pushed away from its inner surface. In other words, the indenter is now a containing wall that traps the particles inside it. If the radius shrinks over time, it will squeeze the particles.

The `units` keyword determines the meaning of the distance units used to define the indenter geometry. A `box` value selects standard distance units as defined by the `units` command, e.g. Angstroms for units = real or metal. A `lattice` value means the distance units are in lattice spacings. The `lattice` command must have been previously used to define the lattice spacing. The (x,y,z) coords of the indenter position are scaled by the x,y,z lattice spacings respectively. The radius of a spherical or cylindrical indenter is scaled by the x lattice spacing.

Note that the `units` keyword only affects indenter geometry parameters specified directly with numbers, not those specified as variables. In the latter case, you should use the `xlat`, `ylat`, `zlat` keywords of the `thermo_style` command if you want to include lattice spacings in a variable formula.

The force constant `K` is not affected by the `units` keyword. It is always in force/distance² units where force and distance are defined by the `units` command. If you wish `K` to be scaled by the lattice spacing, you can define `K` with a variable whose formula contains `xlat`, `ylat`, `zlat` keywords of the `thermo_style` command, e.g.

```
variable k equal 100.0/xlat/xlat
fix 1 all indent $k sphere ...
```

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

No information about this fix is written to *binary restart files*.

The `fix_modify energy` option is supported by this fix to add the energy of interaction between atoms and the indenter to the global potential energy of the system as part of *thermodynamic output*. The default setting for this fix is `fix_modify energy no`. The energy of each particle interacting with the indenter is $K/3 (r - R)^3$.

The `fix_modify respa` option is supported by this fix. This allows to set at which level of the `r-RESPA` integrator the fix is adding its forces. Default is the outermost level.

This fix computes a global scalar energy and a global 3-vector of forces (on the indenter), which can be accessed by various *output commands*. The scalar and vector values calculated by this fix are “extensive”.

The forces due to this fix are imposed during an energy minimization, invoked by the `minimize` command. Note that if you define the indenter geometry with a variable using a time-dependent formula, LAMMPS uses the iteration count in the minimizer as the timestep. But it is almost certainly a bad idea to have the indenter change its position or size during a minimization. LAMMPS does not check if you have done this.

Note

If you want the atom/indenter interaction energy to be included in the total potential energy of the system (the quantity being minimized), you must enable the *fix_modify energy* option for this fix.

2.81.5 Restrictions

none

2.81.6 Related commands

none

2.81.7 Default

The option defaults are side = out and units = lattice.

2.82 fix ipi command

2.82.1 Syntax

```
fix ID group-ID ipi address port [unix] [reset]
```

- ID, group-ID are documented in *fix* command
- ipi = style name of this fix command
- address = internet address (FQDN or IP), or UNIX socket name
- port = port number (ignored for UNIX sockets)
- zero or more keywords may be appended
- keyword = *unix* or *reset*

unix args = none = use a unix socket

reset args = none = reset electrostatics at each call

2.82.2 Examples

```
fix 1 all ipi my.server.com 12345
fix 1 all ipi mysocket 666 unix reset
```

2.82.3 Description

This fix enables LAMMPS to be run as a client for the i-PI Python wrapper ([IPI](#)). i-PI is a universal force engine, designed to perform advanced molecular simulations, with a special focus on path integral molecular dynamics (PIMD) simulation. The philosophy behind i-PI is to separate the evaluation of the energy and forces, which is delegated to the client, and the evolution of the dynamics, that is the responsibility of i-PI. This approach also simplifies combining energies computed from different codes, which can for instance be useful to mix first-principles calculations, empirical force fields or machine-learning potentials. The following publication ([IPI-CPC-2014](#)) discusses the overall implementation of i-PI, and focuses on path-integral techniques, while a later release ([IPI-CPC-2019](#)) introduces several additional features and simulation schemes.

The communication between i-PI and LAMMPS takes place using sockets, and is reduced to the bare minimum. All the parameters of the dynamics are specified in the input of i-PI, and all the parameters of the force field must be specified as LAMMPS inputs, preceding the *fix ipi* command.

The server address must be specified by the *address* argument, and can be either the IP address, the fully-qualified name of the server, or the name of a UNIX socket for local, faster communication. In the case of internet sockets, the *port* argument specifies the port number on which i-PI is listening, while the *unix* optional switch specifies that the socket is a UNIX socket.

Note that there is no check of data integrity, or that the atomic configurations make sense. It is assumed that the species in the i-PI input are listed in the same order as in the data file of LAMMPS. The initial configuration is ignored, as it will be substituted with the coordinates received from i-PI before forces are ever evaluated.

A note of caution when using potentials that contain long-range electrostatics, or that contain parameters that depend on box size: all of these options will be initialized based on the cell size in the LAMMPS-side initial configuration and kept constant during the run. This is required to e.g. obtain reproducible and conserved forces. If the cell varies too wildly, it may be advisable to re-initialize these interactions at each call. This behavior can be requested by setting the *reset* switch.

2.82.4 Obtaining i-PI

Here are the commands to set up a virtual environment and install i-PI into it with all its dependencies via the PyPI repository and the pip package manager.

```
python -m venv ipienv
source ipienv/bin/activate
pip install --upgrade pip
pip install ipi
```

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

There is no restart information associated with this fix, since all the dynamical parameters are dealt with by i-PI.

2.82.6 Restrictions

Using this fix on anything other than all atoms requires particular care, since i-PI will know nothing on atoms that are not those whose coordinates are transferred. However, one could use this strategy to define an external potential acting on the atoms that are moved by i-PI.

Since the i-PI code uses atomic units internally, this fix needs to convert LAMMPS data to and from its *specified units* accordingly when communicating with i-PI. This is not possible for reduced units (“units lj”) and thus *fix ipi* will stop with an error in this case.

This fix is part of the MISC package. It is only enabled if LAMMPS was built with that package. See the *Build package* page for more info. Because of the use of UNIX domain sockets, this fix will only work in a UNIX environment.

2.82.7 Related commands

fix nve

(IPI-CPC-2014) Ceriotti, More and Manolopoulos, Comp Phys Comm 185, 1019-1026 (2014).

(IPI-CPC-2019) Kapil et al., Comp Phys Comm 236, 214-223 (2019).

(IPI) <https://ipi-code.org>

2.83 fix langevin command

Accelerator Variants: *langevin/kk*

2.83.1 Syntax

```
fix ID group-ID langevin Tstart Tstop damp seed keyword values ...
```

- ID, group-ID are documented in *fix* command
- langevin = style name of this fix command
- Tstart,Tstop = desired temperature at start/end of run (temperature units)
- Tstart can be a variable (see below)
- damp = damping parameter (time units)
- seed = random number seed to use for white noise (positive integer)
- zero or more keyword/value pairs may be appended
- keyword = *angmom* or *gif* or *omega* or *scale* or *tally* or *zero*

angmom value = no or factor
 no = do not thermostat rotational degrees of freedom via the angular momentum
 factor = do thermostat rotational degrees of freedom via the angular momentum and apply [_](#)
 ↵ numeric scale factor as discussed below
 gjf value = no or vfull or vhalf
 no = use standard formulation
 vfull = use Gronbech-Jensen/Farago formulation
 vhalf = use 2GJ formulation
 omega value = no or yes
 no = do not thermostat rotational degrees of freedom via the angular velocity
 yes = do thermostat rotational degrees of freedom via the angular velocity
 scale values = type ratio
 type = atom type (1-N)
 ratio = factor by which to scale the damping coefficient
 tally value = no or yes
 no = do not tally the energy added/subtracted to atoms
 yes = do tally the energy added/subtracted to atoms
 zero value = no or yes
 no = do not set total random force to zero
 yes = set total random force to zero

2.83.2 Examples

```
fix 3 boundary langevin 1.0 1.0 1000.0 699483
fix 1 all langevin 1.0 1.1 100.0 48279 scale 3 1.5
fix 1 all langevin 1.0 1.1 100.0 48279 angmom 3.333
```

2.83.3 Description

Apply a Langevin thermostat as described in ([Schneider](#)) to a group of atoms which models an interaction with a background implicit solvent. Used with [fix nve](#), this command performs Brownian dynamics (BD), since the total force on each atom will have the form:

$$\begin{aligned} F &= F_c + F_f + F_r \\ F_f &= -\frac{m}{\text{damp}}v \\ F_r &\propto \sqrt{\frac{k_B T m}{dt \text{ damp}}} \end{aligned}$$

F_c is the conservative force computed via the usual inter-particle interactions ([pair_style](#), [bond_style](#), etc). The F_f and F_r terms are added by this fix on a per-particle basis. See the [pair_style dpd/tstat](#) command for a thermostating option that adds similar terms on a pairwise basis to pairs of interacting particles.

F_f is a frictional drag or viscous damping term proportional to the particle's velocity. The proportionality constant for each atom is computed as $\frac{m}{\text{damp}}$, where m is the mass of the particle and damp is the damping factor specified by the user.

F_r is a force due to solvent atoms at a temperature T randomly bumping into the particle. As derived from the fluctuation/dissipation theorem, its magnitude as shown above is proportional to $\sqrt{\frac{k_B T m}{dt \text{ damp}}}$, where k_B is the Boltzmann constant, T is the desired temperature, m is the mass of the particle, dt is the timestep size, and damp is the damping factor. Random numbers are used to randomize the direction and magnitude of this force as described in ([Dunweg](#)), where a uniform random number is used (instead of a Gaussian random number) for speed.

Note that unless you use the *omega* or *angmom* keywords, the thermostat effect of this fix is applied to only the translational degrees of freedom for the particles, which is an important consideration for finite-size particles, which have rotational degrees of freedom, are being thermostatted. The translational degrees of freedom can also have a bias velocity removed from them before thermostating takes place; see the description below.

Note

Unlike the [fix nvt](#) command which performs Nose/Hoover thermostatting AND time integration, this fix does NOT perform time integration. It only modifies forces to effect thermostatting. Thus you must use a separate time integration fix, like [fix nve](#) to actually update the velocities and positions of atoms using the modified forces. Likewise, this fix should not normally be used on atoms that also have their temperature controlled by another fix - e.g. by [fix nvt](#) or [fix temp/rescale](#) commands.

See the [Howto thermostat](#) page for a discussion of different ways to compute temperature and perform thermostatting.

The desired temperature at each timestep is a ramped value during the run from *Tstart* to *Tstop*.

Tstart can be specified as an equal-style or atom-style *variable*. In this case, the *Tstop* setting is ignored. 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 target temperature.

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 temperature.

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 temperature with optional time-dependence as well.

Like other fixes that perform thermostatting, this fix can be used with [compute commands](#) that remove a “bias” from the atom velocities. E.g. to apply the thermostat only to atoms within a spatial *region*, or to remove the center-of-mass velocity from a group of atoms, or to remove the x-component of velocity from the calculation.

This is not done by default, but only if the [fix_modify](#) command is used to assign a temperature compute to this fix that includes such a bias term. See the doc pages for individual [compute temp commands](#) to determine which ones include a bias. In this case, the thermostat works in the following manner: bias is removed from each atom, thermostatting is performed on the remaining thermal degrees of freedom, and the bias is added back in.

The *damp* parameter is specified in time units and determines how rapidly the temperature is relaxed. For example, a value of 100.0 means to relax the temperature in a timespan of (roughly) 100 time units (τ or fs or ps - see the [units](#) command). The damp factor can be thought of as inversely related to the viscosity of the solvent. I.e. a small relaxation time implies a high-viscosity solvent and vice versa. See the discussion about γ and viscosity in the documentation for the [fix viscous](#) command for more details.

The random # *seed* must be a positive integer. A Marsaglia random number generator is used. Each processor uses the input seed to generate its own unique seed and its own stream of random numbers. Thus the dynamics of the system will not be identical on two runs on different numbers of processors.

The keyword/value option pairs are used in the following ways.

The keyword *angmom* and *omega* keywords enable thermostatting of rotational degrees of freedom in addition to the usual translational degrees of freedom. This can only be done for finite-size particles.

A simulation using atom_style sphere defines an omega for finite-size spheres. A simulation using atom_style ellipsoid defines a finite size and shape for aspherical particles and an angular momentum. The Langevin formulas for thermostatting the rotational degrees of freedom are the same as those above, where force is replaced by torque, m is

replaced by the moment of inertia I , and v is replaced by ω (which is derived from the angular momentum in the case of aspherical particles).

The rotational temperature of the particles can be monitored by the [compute temp/sphere](#) and [compute temp/asphere](#) commands with their rotate options.

For the *omega* keyword there is also a scale factor of $\frac{10.0}{3.0}$ that is applied as a multiplier on the F_f (damping) term in the equation above and of $\sqrt{\frac{10.0}{3.0}}$ as a multiplier on the F_r term. This does not affect the thermostating behavior of the Langevin formalism but ensures that the randomized rotational diffusivity of spherical particles is correct.

For the *angmom* keyword a similar scale factor is needed which is $\frac{10.0}{3.0}$ for spherical particles, but is anisotropic for aspherical particles (e.g. ellipsoids). Currently LAMMPS only applies an isotropic scale factor, and you can choose its magnitude as the specified value of the *angmom* keyword. If your aspherical particles are (nearly) spherical than a value of $\frac{10.0}{3.0} = 3.\bar{3}$ is a good choice. If they are highly aspherical, a value of 1.0 is as good a choice as any, since the effects on rotational diffusivity of the particles will be incorrect regardless. Note that for any reasonable scale factor, the thermostating effect of the *angmom* keyword on the rotational temperature of the aspherical particles should still be valid.

The keyword *scale* allows the damp factor to be scaled up or down by the specified factor for atoms of that type. This can be useful when different atom types have different sizes or masses. It can be used multiple times to adjust damp for several atom types. Note that specifying a ratio of 2 increases the relaxation time which is equivalent to the solvent's viscosity acting on particles with $\frac{1}{2}$ the diameter. This is the opposite effect of scale factors used by the [fix viscous](#) command, since the damp factor in [fix langevin](#) is inversely related to the γ factor in [fix viscous](#). Also note that the damping factor in [fix langevin](#) includes the particle mass in F_f , unlike [fix viscous](#). Thus the mass and size of different atom types should be accounted for in the choice of ratio values.

The keyword *tally* enables the calculation of the cumulative energy added/subtracted to the atoms as they are thermostatted. Effectively it is the energy exchanged between the infinite thermal reservoir and the particles. As described below, this energy can then be printed out or added to the potential energy of the system to monitor energy conservation.

Note

This accumulated energy does NOT include kinetic energy removed by the *zero* flag. LAMMPS will print a warning when both options are active.

The keyword *zero* can be used to eliminate drift due to the thermostat. Because the random forces on different atoms are independent, they do not sum exactly to zero. As a result, this fix applies a small random force to the entire system, and the center-of-mass of the system undergoes a slow random walk. If the keyword *zero* is set to *yes*, the total random force is set exactly to zero by subtracting off an equal part of it from each atom in the group. As a result, the center-of-mass of a system with zero initial momentum will not drift over time.

The keyword *gif* can be used to run the [Gronbech-Jensen/Farago](#) time-discretization of the Langevin model. As described in the papers cited below, the purpose of this method is to enable longer timesteps to be used (up to the numerical stability limit of the integrator), while still producing the correct Boltzmann distribution of atom positions.

The current implementation provides the user with the option to output the velocity in one of two forms: *vfull* or *vhalf*, which replaces the outdated option *yes*. The *gif* option *vfull* outputs the on-site velocity given in [Gronbech-Jensen/Farago](#); this velocity is shown to be systematically lower than the target temperature by a small amount, which grows quadratically with the timestep. The *gif* option *vhalf* outputs the 2GJ half-step velocity given in [Gronbech Jensen/Gronbech-Jensen](#); for linear systems, this velocity is shown to not have any statistical errors for any stable time step. An overview of statistically correct Boltzmann and Maxwell-Boltzmann sampling of true on-site and true half-step velocities is given in [Gronbech-Jensen](#). Regardless of the choice of output velocity, the sampling of the configurational distribution of atom positions is the same, and linearly consistent with the target temperature.

Styles with a *gpu*, *intel*, *kk*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix.

They have been optimized to run faster, depending on your available hardware, as discussed on the [Accelerator packages](#) page. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the GPU, INTEL, KOKKOS, OPENMP, and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the [Build package](#) page for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the [-suffix command-line switch](#) when you invoke LAMMPS, or you can use the [suffix](#) command in your input script.

See the [Accelerator packages](#) page for more instructions on how to use the accelerated styles effectively.

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

No information about this fix is written to [binary restart files](#). Because the state of the random number generator is not saved in restart files, this means you cannot do “exact” restarts with this fix, where the simulation continues on the same as if no restart had taken place. However, in a statistical sense, a restarted simulation should produce the same behavior.

The [fix_modify temp](#) option is supported by this fix. You can use it to assign a temperature [compute](#) you have defined to this fix which will be used in its thermostating procedure, as described above. For consistency, the group used by this fix and by the compute should be the same.

The cumulative energy change in the system imposed by this fix is included in the [thermodynamic output](#) keywords [ecouple](#) and [econserve](#), but only if the [tally](#) keyword is set to yes. See the [thermo_style](#) page for details.

This fix computes a global scalar which can be accessed by various [output commands](#). The scalar is the same cumulative energy change due to this fix described in the previous paragraph. The scalar value calculated by this fix is “extensive”. Note that calculation of this quantity also requires setting the [tally](#) keyword to yes.

This fix can ramp its target temperature over multiple runs, using the [start](#) and [stop](#) keywords of the [run](#) command. See the [run](#) command for details of how to do this.

This fix is not invoked during [energy minimization](#).

2.83.5 Restrictions

For [gif](#) do not choose damp=dt/2. [gif](#) is not compatible with [run_style respa](#).

2.83.6 Related commands

[fix nvt](#), [fix temp/rescale](#), [fix viscous](#), [fix nvt](#), [pair_style dpd/tstat](#)

2.83.7 Default

The option defaults are angmom = no, omega = no, scale = 1.0 for all types, tally = no, zero = no, gif = no.

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

(Schneider) Schneider and Stoll, Phys Rev B, 17, 1302 (1978).

(Gronbech-Jensen) Gronbech-Jensen and Farago, Mol Phys, 111, 983 (2013); Gronbech-Jensen, Hayre, and Farago, Comp Phys Comm, 185, 524 (2014)

(Gronbech-Jensen) Gronbech Jensen and Gronbech-Jensen, Mol Phys, 117, 2511 (2019)

(Gronbech-Jensen) Gronbech-Jensen, Mol Phys (2019); <https://doi.org/10.1080/00268976.2019.1662506>

2.84 fix langevin/drude command

2.84.1 Syntax

```
fix ID group-ID langevin/drude Tcom damp_com seed_com Tdrude damp_drude seed_drude keyword
  ↘values ...
```

- ID, group-ID are documented in [fix](#) command
- langevin/drude = style name of this fix command
- Tcom = desired temperature of the centers of mass (temperature units)
- damp_com = damping parameter for the thermostat on centers of mass (time units)
- seed_com = random number seed to use for white noise of the thermostat on centers of mass (positive integer)
- Tdrude = desired temperature of the Drude oscillators (temperature units)
- damp_drude = damping parameter for the thermostat on Drude oscillators (time units)
- seed_drude = random number seed to use for white noise of the thermostat on Drude oscillators (positive integer)
- zero or more keyword/value pairs may be appended
- keyword = *zero*
zero value = no or yes
no = do not set total random force on centers of mass to zero
yes = set total random force on centers of mass to zero

2.84.2 Examples

```
fix 3 all langevin/drude 300.0 100.0 19377 1.0 20.0 83451
fix 1 all langevin/drude 298.15 100.0 19377 5.0 10.0 83451 zero yes
```

Example input scripts available: examples/PACKAGES/drude

2.84.3 Description

Apply two Langevin thermostats as described in ([Jiang1](#)) for thermalizing the reduced degrees of freedom of Drude oscillators. This link describes how to use the [thermalized Drude oscillator model](#) in LAMMPS and polarizable models in LAMMPS are discussed on the [Howto polarizable](#) doc page.

Drude oscillators are a way to simulate polarizable atoms, by splitting them into a core and a Drude particle bound by a harmonic bond. The thermalization works by transforming the particles degrees of freedom by these equations. In these equations upper case denotes atomic or center of mass values and lower case denotes Drude particle or dipole values. Primes denote the transformed (reduced) values, while bare letters denote the original values.

Velocities:

$$V' = \frac{MV + mv}{M'}$$

$$v' = v - V$$

Masses:

$$M' = M + m$$

$$m' = \frac{M m}{M'}$$

The Langevin forces are computed as

$$F' = -\frac{M'}{\text{damp}_{\text{com}}} V' + F'_r$$

$$f' = -\frac{m'}{\text{damp}_{\text{drude}}} v' + f'_r$$

F'_r is a random force proportional to $\sqrt{\frac{2k_B T_{\text{com}} m'}{\text{damp}_{\text{com}}}}$. f'_r is a random force proportional to $\sqrt{\frac{2k_B T_{\text{drude}} m'}{\text{damp}_{\text{drude}}}}$. Then the real forces acting on the particles are computed from the inverse transform:

$$F = \frac{M}{M'} F' - f'$$

$$f = \frac{m}{M'} F' + f'$$

This fix also thermostats non-polarizable atoms in the group at temperature T_{com} , as if they had a massless Drude partner. The Drude particles themselves need not be in the group. The center of mass and the dipole are thermostatted iff the core atom is in the group.

Note that the thermostat effect of this fix is applied to only the translational degrees of freedom of the particles, which is an important consideration if finite-size particles, which have rotational degrees of freedom, are being thermostatted. The translational degrees of freedom can also have a bias velocity removed from them before thermostatting takes place; see the description below.

Note

Like the [fix langevin](#) command, this fix does NOT perform time integration. It only modifies forces to effect thermostatting. Thus you must use a separate time integration fix, like [fix nve](#) or [fix nph](#) to actually update the velocities and positions of atoms using the modified forces. Likewise, this fix should not normally be used on atoms that also have their temperature controlled by another fix - e.g. by [fix nvt](#) or [fix temp/rescale](#) commands.

See the [Howto thermostat](#) page for a discussion of different ways to compute temperature and perform thermostatting.

This fix requires each atom know whether it is a Drude particle or not. You must therefore use the [fix drude](#) command to specify the Drude status of each atom type.

Note

only the Drude core atoms need to be in the group specified for this fix. A Drude electron will be transformed together with its cores even if it is not itself in the group. It is safe to include Drude electrons or non-polarizable atoms in the group. The non-polarizable atoms will simply be thermostatted as if they had a massless Drude partner (electron).

 Note

Ghost atoms need to know their velocity for this fix to act correctly. You must use the `comm_modify` command to enable this, e.g.

```
comm_modify vel yes
```

T_{com} is the target temperature of the centers of mass, which would be used to thermostat the non-polarizable atoms. T_{drude} is the (normally low) target temperature of the core-Drude particle pairs (dipoles). T_{com} and T_{drude} 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 target temperature.

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 temperature.

Like other fixes that perform thermostating, this fix can be used with `compute commands` that remove a “bias” from the atom velocities. E.g. to apply the thermostat only to atoms within a spatial `region`, or to remove the center-of-mass velocity from a group of atoms, or to remove the x-component of velocity from the calculation.

This is not done by default, but only if the `fix_modify` command is used to assign a temperature compute to this fix that includes such a bias term. See the doc pages for individual `compute temp commands` to determine which ones include a bias. In this case, the thermostat works in the following manner: bias is removed from each atom, thermostating is performed on the remaining thermal degrees of freedom, and the bias is added back in.

Note: The temperature thermostating the core-Drude particle pairs should be chosen low enough, so as to mimic as closely as possible the self-consistent minimization. It must however be high enough, so that the dipoles can follow the local electric field exerted by the neighboring atoms. The optimal value probably depends on the temperature of the centers of mass and on the mass of the Drude particles.

$damp_{com}$ is the characteristic time for reaching thermal equilibrium of the centers of mass. For example, a value of 100.0 means to relax the temperature of the centers of mass in a timespan of (roughly) 100 time units (`tau` or `fs` or `ps` - see the `units` command). $damp_{drude}$ is the characteristic time for reaching thermal equilibrium of the dipoles. It is typically a few timesteps.

The number $seed_{com}$ and $seed_{drude}$ are positive integers. They set the seeds of the Marsaglia random number generators used for generating the random forces on centers of mass and on the dipoles. Each processor uses the input seed to generate its own unique seed and its own stream of random numbers. Thus the dynamics of the system will not be identical on two runs on different numbers of processors.

The keyword `zero` can be used to eliminate drift due to the thermostat on centers of mass. Because the random forces on different centers of mass are independent, they do not sum exactly to zero. As a result, this fix applies a small random force to the entire system, and the momentum of the total center of mass of the system undergoes a slow random walk. If the keyword `zero` is set to `yes`, the total random force on the centers of mass is set exactly to zero by subtracting off an equal part of it from each center of mass in the group. As a result, the total center of mass of a system with zero initial momentum will not drift over time.

The actual temperatures of cores and Drude particles, in center-of-mass and relative coordinates, respectively, can be calculated using the `compute temp/drude` command.

Usage example for rigid bodies in the NPT ensemble:

```

comm_modify vel yes
fix TEMP all langevin/drude 300. 100. 1256 1. 20. 13977 zero yes
fix NPH ATOMS rigid/nph/small molecule iso 1. 1. 500.
fix NVE DRUDES nve
compute TDRUDE all temp/drude
thermo_style custom step cpu etotal ke pe ebond ecoul elong press vol temp c_TDRUDE[1] c_
→ TDRUDE[2]

```

Comments:

- Drude particles should not be in the rigid group, otherwise the Drude oscillators will be frozen and the system will lose its polarizability.
- *zero yes* avoids a drift of the center of mass of the system, but is a bit slower.
- Use two different random seeds to avoid unphysical correlations.
- Temperature is controlled by the fix *langevin/drude*, so the time-integration fixes do not thermostat. Don't forget to time-integrate both cores and Drude particles.
- Pressure is time-integrated only once by using *nve* for Drude particles and *nph* for atoms/cores (or vice versa). Do not use *nph* for both.
- The temperatures of cores and Drude particles are calculated by *compute temp/drude*
- Contrary to the alternative thermostating using Nose-Hoover thermostat fix *npt* and *fix drude/transform*, the *fix_modify* command is not required here, because the fix *nph* computes the global pressure even if its group is *ATOMS*. This is what we want. If we thermostatted *ATOMS* using *npt*, the pressure should be the global one, but the temperature should be only that of the cores. That's why the command *fix_modify* should be called in that case.

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

No information about this fix is written to *binary restart files*. Because the state of the random number generator is not saved in restart files, this means you cannot do “exact” restarts with this fix, where the simulation continues on the same as if no restart had taken place. However, in a statistical sense, a restarted simulation should produce the same behavior.

The *fix_modify temp* option is supported by this fix. You can use it to assign a temperature *compute* you have defined to this fix which will be used in its thermostating procedure, as described above. For consistency, the group used by the compute should include the group of this fix and the Drude particles.

This fix is not invoked during *energy minimization*.

2.84.5 Restrictions

none

2.84.6 Related commands

fix langevin, fix drude, fix drude/transform, compute temp/drude, pair_style thole

2.84.7 Default

The option defaults are zero = no.

(Jiang1) Jiang, Hardy, Phillips, MacKerell, Schulten, and Roux, J Phys Chem Lett, 2, 87-92 (2011).

2.85 fix langevin/eff command

2.85.1 Syntax

```
fix ID group-ID langevin/eff Tstart Tstop damp seed keyword values ...
```

- ID, group-ID are documented in *fix* command
- langevin/eff = style name of this fix command
- Tstart,Tstop = desired temperature at start/end of run (temperature units)
- damp = damping parameter (time units)
- seed = random number seed to use for white noise (positive integer)
- zero or more keyword/value pairs may be appended

keyword = scale or tally or zero

scale values = type ratio

type = atom type (1-N)

ratio = factor by which to scale the damping coefficient

tally values = no or yes

no = do not tally the energy added/subtracted to atoms

yes = do tally the energy added/subtracted to atoms

zero value = no or yes

no = do not set total random force to zero

yes = set total random force to zero

2.85.2 Examples

```
fix 3 boundary langevin/eff 1.0 1.0 10.0 699483
fix 1 all langevin/eff 1.0 1.1 10.0 48279 scale 3 1.5
```

2.85.3 Description

Apply a Langevin thermostat as described in ([Schneider](#)) to a group of nuclei and electrons in the [electron force field](#) model. Used with [fix nve/eff](#), this command performs Brownian dynamics (BD), since the total force on each atom will have the form:

$$\begin{aligned} F &= F_c + F_f + F_r \\ F_f &= -\frac{m}{\text{damp}} v \\ F_r &\propto \sqrt{\frac{k_B T m}{dt \text{ damp}}} \end{aligned}$$

F_c is the conservative force computed via the usual inter-particle interactions ([pair_style](#)). The F_f and F_r terms are added by this fix on a per-particle basis.

The operation of this fix is exactly like that described by the [fix langevin](#) command, except that the thermostating is also applied to the radial electron velocity for electron particles.

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

No information about this fix is written to [binary restart files](#). Because the state of the random number generator is not saved in restart files, this means you cannot do “exact” restarts with this fix, where the simulation continues on the same as if no restart had taken place. However, in a statistical sense, a restarted simulation should produce the same behavior.

The [fix_modify temp](#) option is supported by this fix. You can use it to assign a temperature [compute](#) you have defined to this fix which will be used in its thermostating procedure, as described above. For consistency, the group used by this fix and by the compute should be the same.

The cumulative energy change in the system imposed by this fix is included in the [thermodynamic output](#) keywords [ecouple](#) and [econserve](#), but only if the [tally](#) keyword is set to yes. See the [thermo_style](#) page for details.

This fix computes a global scalar which can be accessed by various [output commands](#). The scalar is the same cumulative energy change due to this fix described in the previous paragraph. The scalar value calculated by this fix is “extensive”. Note that calculation of this quantity also requires setting the [tally](#) keyword to yes.

This fix can ramp its target temperature over multiple runs, using the [start](#) and [stop](#) keywords of the [run](#) command. See the [run](#) command for details of how to do this.

This fix is not invoked during [energy minimization](#).

2.85.5 Restrictions

none

This fix is part of the EFF package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

2.85.6 Related commands

fix langevin

2.85.7 Default

The option defaults are scale = 1.0 for all types and tally = no.

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

(Schneider) Schneider and Stoll, Phys Rev B, 17, 1302 (1978).

2.86 fix langevin/spin command

2.86.1 Syntax

`fix ID group-ID langevin/spin T Tdamp seed`

- ID, group-ID are documented in [fix](#) command
- langevin/spin = style name of this fix command
- T = desired temperature of the bath (temperature units, K in metal units)
- Tdamp = transverse magnetic damping parameter (adim)
- seed = random number seed to use for white noise (positive integer)

2.86.2 Examples

`fix 2 all langevin/spin 300.0 0.01 21`

2.86.3 Description

Apply a Langevin thermostat as described in ([Mayergoyz](#)) to the magnetic spins associated to the atoms. Used with [fix nve/spin](#), this command performs Brownian dynamics (BD). A random torque and a transverse dissipation are applied to each spin *i* according to the following stochastic differential equation:

$$\frac{d\vec{s}_i}{dt} = \frac{1}{(1 + \lambda^2)} ((\vec{\omega}_i + \vec{\eta}) \times \vec{s}_i + \lambda \vec{s}_i \times (\vec{\omega}_i \times \vec{s}_i))$$

with λ the transverse damping, and η a random vector. This equation is referred to as the stochastic Landau-Lifshitz (sLL) equation.

The components of η are drawn from a Gaussian probability law. Their amplitude is defined as a proportion of the temperature of the external thermostat T (in K in metal units).

More details about this implementation are reported in ([Tranchida](#)).

Note: due to the form of the sLL equation, this fix has to be defined just before the nve/spin fix (and after all other magnetic fixes). As an example:

```
fix 1 all precession/spin zeeman 0.01 0.0 0.0 1.0
fix 2 all langevin/spin 300.0 0.01 21
fix 3 all nve/spin lattice moving
```

is correct, but defining a force/spin command after the langevin/spin command would give an error message.

Note: The random # *seed* must be a positive integer. A Marsaglia random number generator is used. Each processor uses the input seed to generate its own unique seed and its own stream of random numbers. Thus the dynamics of the system will not be identical on two runs on different numbers of processors.

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

No information about this fix is written to *binary restart files*. Because the state of the random number generator is not saved in restart files, this means you cannot do “exact” restarts with this fix, where the simulation continues on the same as if no restart had taken place. However, in a statistical sense, a restarted simulation should produce the same behavior.

This fix is not invoked during *energy minimization*.

2.86.5 Restrictions

The *langevin/spin* fix is part of the SPIN package. This style is only enabled if LAMMPS was built with this package. See the *Build package* page for more info.

The numerical integration has to be performed with *fix nve/spin* when *fix langevin/spin* is enabled.

This fix has to be the last defined magnetic fix before the time integration fix (e.g. *fix nve/spin*).

2.86.6 Related commands

fix nve/spin, *fix precession/spin*

2.86.7 Default

none

(**Mayergoyz**) I.D. Mayergoyz, G. Bertotti, C. Serpico (2009). Elsevier (2009)

(**Tranchida**) Tranchida, Plimpton, Thibaudeau and Thompson, Journal of Computational Physics, 372, 406-425, (2018).

2.87 fix lb/fluid command

2.87.1 Syntax

```
fix ID group-ID lb/fluid nevery viscosity density keyword values ...
```

- ID, group-ID are documented in [fix](#) command
- lb/fluid = style name of this fix command
- nevery = update the lattice-Boltzmann fluid every this many timesteps (should normally be 1)
- viscosity = the fluid viscosity (units of mass/(time*length)).
- density = the fluid density.
- zero or more keyword/value pairs may be appended

keyword = *dx* or *dm* or *noise* or *stencil* or *read_restart* or *write_restart* or *zwall_velocity* or *pressurebcx* or *bodyforce* or *D3Q19* or *dumpxdmf* or *linearInit* or *dof* or *scaleGamma* or *a0* or *npits* or *wp* or *sw*

dx values = *dx_LB* = the lattice spacing.

dm values = *dm_LB* = the lattice-Boltzmann mass unit.

noise values = Temperature seed

Temperature = fluid temperature.

seed = random number generator seed (positive integer)

stencil values = 2 (trilinear stencil, the default), 3 (3-point immersed boundary stencil), or 4
 \rightarrow (4-point Keys' interpolation stencil)

read_restart values = restart file = name of the restart file to use to restart a fluid run.

write_restart values = N = write a restart file every N MD timesteps.

zwall_velocity values = velocity_bottom velocity_top = velocities along the y-direction of the
 \rightarrow bottom and top walls (located at z=zmin and z=zmax).

pressurebcx values = *pgradav* = imposes a pressure jump at the (periodic) x-boundary of
 \rightarrow *pgradav***Lx**1000.

bodyforce values = *bodyforcex* *bodyforcey* *bodyforcez* = the x,y and z components of a constant
 \rightarrow body force added to the fluid.

D3Q19 values = none (used to switch from the default D3Q15, 15 velocity lattice, to the D3Q19, 19
 \rightarrow velocity lattice).

dumpxdmf values = N file timeI

N = output the force and torque every N timesteps

file = output file name

timeI = 1 (use simulation time to index xdmf file), 0 (use output frame number to index xdmf_file)

linearInit values = none = initialize density and velocity using linear interpolation (default is
 \rightarrow uniform density, no velocities)

dof values = dof = specify the number of degrees of freedom for temperature calculation

scaleGamma values = type *gammaFactor*

type = atom type (1-N)

gammaFactor = factor to scale the setGamma gamma value by, for the specified atom type.

a0 values = *a_0_real* = the square of the speed of sound in the fluid.

npits values = *npits h_p l_p l_pp l_e*

npits = number of pit regions

h_p = z-height of pit regions (floor to bottom of slit)

l_p = x-length of pit regions

l_pp = x-length of slit regions between consecutive pits

l_e = x-length of slit regions at ends

wp values = w_p = y-width of slit regions (defaults to full width if not present or if sw active)
 sw values = none (turns on y-sidewalls (in xz plane) if npits option active)

2.87.2 Examples

```
fix 1 all lb/fluid 1 1.0 0.0009982071 dx 1.2 dm 0.001
fix 1 all lb/fluid 1 1.0 0.0009982071 dx 1.2 dm 0.001 noise 300.0 2761
fix 1 all lb/fluid 1 1.0 1.0 dx 4.0 dm 10.0 dumpxdmf 500 fflow 0 pressurebcx 0.01 npits 2 20 40 5 0 wp 30
```

2.87.3 Description

Changed in version 24Mar2022.

Implement a lattice-Boltzmann fluid on a uniform mesh covering the LAMMPS simulation domain. Note that this fix was updated in 2022 and is not backward compatible with the previous version. If you need the previous version, please download an older version of LAMMPS. The MD particles described by *group-ID* apply a velocity dependent force to the fluid.

The lattice-Boltzmann algorithm solves for the fluid motion governed by the Navier Stokes equations,

$$\begin{aligned}\partial_t \rho + \partial_\beta (\rho u_\beta) &= 0 \\ \partial_t (\rho u_\alpha) + \partial_\beta (\rho u_\alpha u_\beta) &= \partial_\beta \sigma_{\alpha\beta} + F_\alpha + \partial_\beta (\eta_{\alpha\beta\gamma\nu} \partial_\gamma u_\nu)\end{aligned}$$

with,

$$\eta_{\alpha\beta\gamma\nu} = \eta \left[\delta_{\alpha\gamma} \delta_{\beta\nu} + \delta_{\alpha\nu} \delta_{\beta\gamma} - \frac{2}{3} \delta_{\alpha\beta} \delta_{\gamma\nu} \right] + \Lambda \delta_{\alpha\beta} \delta_{\gamma\nu}$$

where ρ is the fluid density, u is the local fluid velocity, σ is the stress tensor, F is a local external force, and η and Λ are the shear and bulk viscosities respectively. Here, we have implemented

$$\sigma_{\alpha\beta} = -P_{\alpha\beta} = -\rho a_0 \delta_{\alpha\beta}$$

with a_0 set to $\frac{1}{3} \frac{dx^2}{dt}$ by default. You should not normally need to change this default.

The algorithm involves tracking the time evolution of a set of partial distribution functions which evolve according to a velocity discretized version of the Boltzmann equation,

$$(\partial_t + e_{i\alpha} \partial_\alpha) f_i = -\frac{1}{\tau} (f_i - f_i^{eq}) + W_i$$

where the first term on the right hand side represents a single time relaxation towards the equilibrium distribution function, and τ is a parameter physically related to the viscosity. On a technical note, we have implemented a 15 velocity model (D3Q15) as default; however, the user can switch to a 19 velocity model (D3Q19) through the use of the *D3Q19* keyword. Physical variables are then defined in terms of moments of the distribution functions,

$$\begin{aligned}\rho &= \sum_i f_i \\ \rho u_\alpha &= \sum_i f_i e_{i\alpha}\end{aligned}$$

Full details of the lattice-Boltzmann algorithm used can be found in [Denniston et al.](#).

The fluid is coupled to the MD particles described by *group-ID* through a velocity dependent force. The contribution to the fluid force on a given lattice mesh site j due to MD particle α is calculated as:

$$\mathbf{F}_{j\alpha} = \gamma (\mathbf{v}_n - \mathbf{u}_f) \zeta_{j\alpha}$$

where \mathbf{v}_n is the velocity of the MD particle, \mathbf{u}_f is the fluid velocity interpolated to the particle location, and γ is the force coupling constant. This force, as with most forces in LAMMPS, and hence the velocities, are calculated at the half-time step. ζ is a weight assigned to the grid point, obtained by distributing the particle to the nearest lattice sites.

The force coupling constant, γ , is calculated according to

$$\gamma = \frac{2m_u m_v}{m_u + m_v} \left(\frac{1}{\Delta t} \right)$$

Here, m_v is the mass of the MD particle, m_u is a representative fluid mass at the particle location, and Δt is the time step. The fluid mass m_u that the MD particle interacts with is calculated internally. This coupling is chosen to constrain the particle and associated fluid velocity to match at the end of the time step. As with other constraints, such as [shake](#), this constraint can remove degrees of freedom from the simulation which are accounted for internally in the algorithm.

Note

While this fix applies the force of the particles on the fluid, it does not apply the force of the fluid to the particles. There is only one option to include this hydrodynamic force on the particles, and that is through the use of the [lb/viscous](#) fix. This fix adds the hydrodynamic force to the total force acting on the particles, after which any of the built-in LAMMPS integrators can be used to integrate the particle motion. If the [lb/viscous](#) fix is NOT used to add the hydrodynamic force to the total force acting on the particles, this physically corresponds to a situation in which an infinitely massive particle is moving through the fluid (since collisions between the particle and the fluid do not act to change the particle's velocity). In this case, setting `scaleGamma` to -1 for the corresponding particle type will explicitly take this limit (of infinite particle mass) in computing the force coupling for the fluid force.

Physical parameters describing the fluid are specified through *viscosity* and *density*. These parameters should all be given in terms of the mass, distance, and time units chosen for the main LAMMPS run, as they are scaled by the LB timestep, lattice spacing, and mass unit, inside the fix.

The `dx` keyword allows the user to specify a value for the LB grid spacing and the `dm` keyword allows the user to specify the LB mass unit. Inside the fix, parameters are scaled by the lattice-Boltzmann timestep, dt_{LB} , grid spacing, dx_{LB} , and mass unit, dm_{LB} . dt_{LB} is set equal to $n\text{every} \cdot dt_{MD}$, where dt_{MD} is the MD timestep. By default, dm_{LB} is set equal to 1.0, and dx_{LB} is chosen so that $\frac{\tau}{dt} = \frac{3\eta dt}{\rho dx^2}$ is approximately equal to 1.

Note

Care must be taken when choosing both a value for dx_{LB} , and a simulation domain size. This fix uses the same subdivision of the simulation domain among processors as the main LAMMPS program. In order to uniformly cover the simulation domain with lattice sites, the lengths of the individual LAMMPS subdomains must all be evenly divisible by dx_{LB} . If the simulation domain size is cubic, with equal lengths in all dimensions, and the default value for dx_{LB} is used, this will automatically be satisfied.

If the `noise` keyword is used, followed by a positive temperature value, and a positive integer random number seed, the thermal LB algorithm of [Adhikari et al.](#) is used.

If the keyword `stencil` is used, the value sets the number of interpolation points used in each direction. For this, the user has the choice between a trilinear stencil (`stencil 2`), which provides a support of 8 lattice sites, or the 3-point immersed boundary method stencil (`stencil 3`), which provides a support of 27 lattice sites, or the 4-point Keys' interpolation stencil (`stencil 4`), which provides a support of 64 lattice sites. The trilinear stencil is the default as it is better suited for simulation of objects close to walls or other objects, due to its smaller support. The 3-point stencil provides smoother motion of the lattice and is suitable for particles not likely to be close to walls or other objects.

If the keyword `write_restart` is used, followed by a positive integer, N, a binary restart file is printed every N LB timesteps. This restart file only contains information about the fluid. Therefore, a LAMMPS restart file should also be written in order to print out full details of the simulation.

Note

When a large number of lattice grid points are used, the restart files may become quite large.

In order to restart the fluid portion of the simulation, the keyword `read_restart` is specified, followed by the name of the binary lb_fluid restart file to be used.

If the `zwall_velocity` keyword is used y-velocities are assigned to the lower and upper walls. This keyword requires the presence of walls in the z-direction. This is set by assigning fixed boundary conditions in the z-direction. If fixed boundary conditions are present in the z-direction, and this keyword is not used, the walls are assumed to be stationary.

If the `pressurebcx` keyword is used, a pressure jump (implemented by a step jump in density) is imposed at the (periodic) x-boundary. The value set specifies what would be the resulting equilibrium average pressure gradient in the x-direction if the system had a constant cross-section (i.e. resistance to flow). It is converted to a pressure jump by multiplication by the system size in the x-direction. As this value should normally be quite small, it is also assumed to be scaled by 1000.

If the `bodyforce` keyword is used, a constant body force is added to the fluid, defined by its x, y and z components.

If the keyword `D3Q19` is used, the 19 velocity (D3Q19) lattice is used by the lattice-Boltzmann algorithm. By default, the 15 velocity (D3Q15) lattice is used.

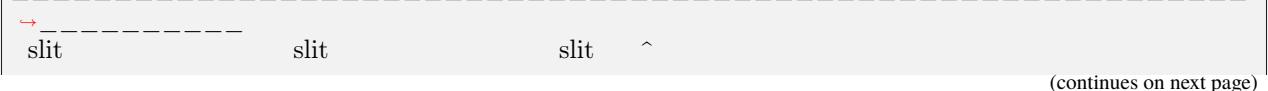
If the `dumpxdmf` keyword is used, followed by a positive integer, N, and a file name, the fluid densities and velocities at each lattice site are output to an xdmf file every N timesteps. This is a binary file format that can be read by visualization packages such as [Paraview](#). The xdmf file format contains a time index for each frame dump and the value `timeI = 1` uses simulation time while 0 uses the output frame number to index xdmf file. The later can be useful if the `dump vtk` command is used to output the particle positions at the same timesteps and you want to visualize both the fluid and particle data together in [Paraview](#).

The `scaleGamma` keyword allows the user to scale the γ value by a factor, `gammaFactor`, for a given atom type. Setting `scaleGamma` to -1 for the corresponding particle type will explicitly take the limit of infinite particle mass in computing the force coupling for the fluid force (see note above).

If the `a0` keyword is used, the value specified is used for the square of the speed of sound in the fluid. If this keyword is not present, the speed of sound squared is set equal to $\frac{1}{3} \left(\frac{dx_{LB}}{dt_{LB}} \right)^2$. Setting $a0 > (\frac{dx_{LB}}{dt_{LB}})^2$ is not allowed, as this may lead to instabilities. As the speed of sound should usually be much larger than any fluid velocity of interest, its value does not normally have a significant impact on the results. As such, it is usually best to use the default for this option.

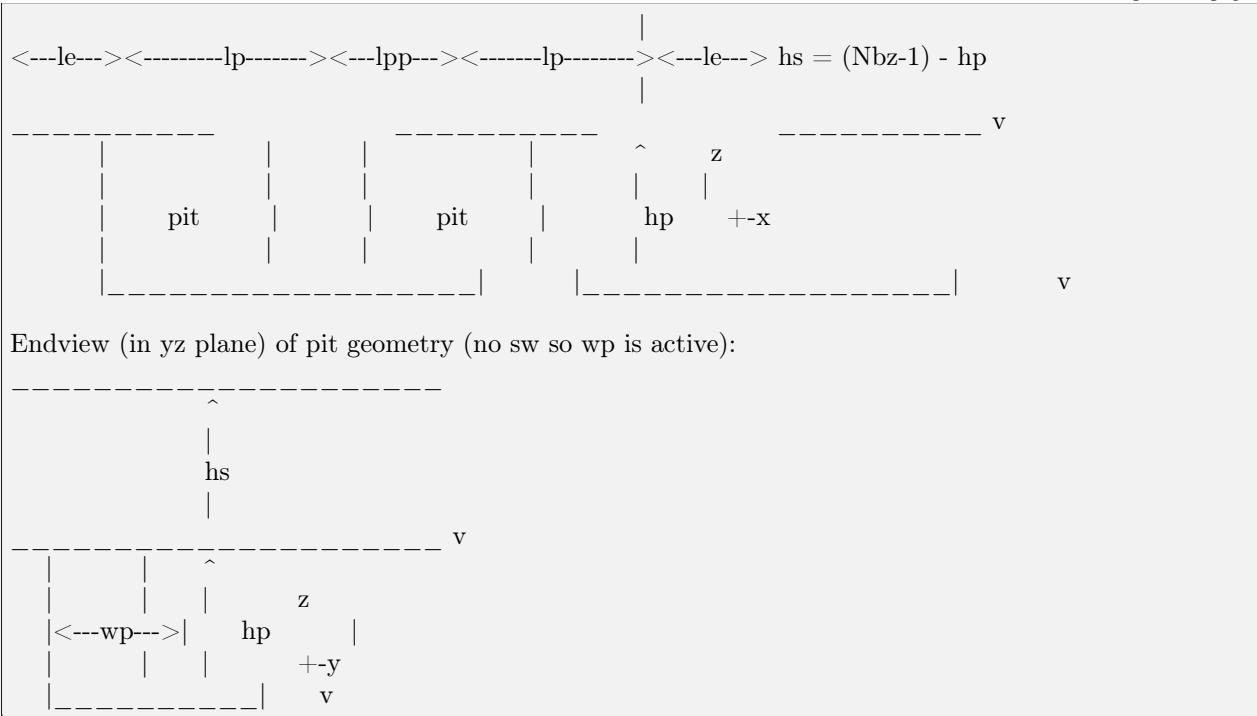
The `npits` keyword (followed by integer arguments: `npits, h_p, l_p, l_pp, l_e`) sets the fluid domain to the pits geometry. These arguments should only be used if you actually want something more complex than a rectangular/cubic geometry. The `npits` value sets the number of pits regions (arranged along x). The remaining arguments are sizes measured in multiples of `dx_lb`: `h_p` is the z-height of the pit regions, `l_p` is the x-length of the pit regions, `l_pp` is the length of the region between consecutive pits (referred to as a “slit” region), and `l_e` is the x-length of the slit regions at each end of the channel. The pit geometry must fill the system in the x-direction but can be longer, in which case it is truncated (which enables asymmetric entrance/exit end sections). The additional `wp` keyword allows the width (in y-direction) of the pit to be specified (the default is full width) and the `sw` keyword indicates that there should be sidewalls in the y-direction (default is periodic in y-direction). These parameters are illustrated below:

Sideview (in xz plane) of pit geometry:



(continues on next page)

(continued from previous page)



For further details, as well as descriptions and results of several test runs, see [Denniston et al.](#). Please include a citation to this paper if the `lb_fluid` fix is used in work contributing to published research.

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

Due to the large size of the fluid data, this fix writes its own binary restart files, if requested, independent of the main LAMMPS [binary restart files](#); no information about `lb_fluid` is written to the main LAMMPS [binary restart files](#).

None of the `fix_modify` options are relevant to this fix.

The fix computes a global scalar which can be accessed by various [output commands](#). The scalar is the current temperature of the group of particles described by *group-ID* along with the fluid constrained to move with them. The temperature is computed via the kinetic energy of the group and fluid constrained to move with them and the total number of degrees of freedom (calculated internally). If the particles are not integrated independently (such as via `fix NVE`) but have additional constraints imposed on them (such as via integration using `fix rigid`) the degrees of freedom removed from these additional constraints will not be properly accounted for. In this case, the user can specify the total degrees of freedom independently using the `dof` keyword.

The fix also computes a global array of values which can be accessed by various [output commands](#). There are 5 entries in the array. The first entry is the temperature of the fluid, the second entry is the total mass of the fluid plus particles, the third through fifth entries give the x, y, and z total momentum of the fluid plus particles.

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](#).

2.87.5 Restrictions

This fix is part of the LATBOLTZ package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

This fix can only be used with an orthogonal simulation domain.

The boundary conditions for the fluid are specified independently to the particles. However, these should normally be specified consistently via the main LAMMPS [boundary](#) command (p p p, p p f, and p f f are the only consistent possibilities). Shrink-wrapped boundary conditions are not permitted with this fix.

This fix must be used before any of [fix lb/viscous](#) and [fix lb/momentum](#) as the fluid needs to be initialized before any of these routines try to access its properties. In addition, in order for the hydrodynamic forces to be added to the particles, this fix must be used in conjunction with the [lb/viscous](#) fix.

This fix needs to be used in conjunction with a standard LAMMPS integrator such as [fix NVE](#) or [fix rigid](#).

2.87.6 Related commands

[fix lb/viscous](#), [fix lb/momentum](#)

2.87.7 Default

dx is chosen such that $\frac{\tau}{dt_{LB}} = \frac{3\eta dt_{LB}}{\rho dx_{LB}^2}$ is approximately equal to 1. dm is set equal to 1.0. $a0$ is set equal to $\frac{1}{3} \left(\frac{dx_{LB}}{dt_{LB}} \right)^2$. The trilinear stencil is used as the default interpolation method. The D3Q15 lattice is used for the lattice-Boltzmann algorithm.

(Denniston et al.) Denniston, C., Afrasiabian, N., Cole-Andre, M.G., Mackay, F. E., Ollila, S.T.T., and Whitehead, T., LAMMPS lb/fluid fix version 2: Improved Hydrodynamic Forces Implemented into LAMMPS through a lattice-Boltzmann fluid, Computer Physics Communications 275 (2022) 108318 .

(Mackay and Denniston) Mackay, F. E., and Denniston, C., Coupling MD particles to a lattice-Boltzmann fluid through the use of conservative forces, J. Comput. Phys. 237 (2013) 289-298.

(Adhikari et al.) Adhikari, R., Stratford, K., Cates, M. E., and Wagner, A. J., Fluctuating lattice Boltzmann, Europhys. Lett. 71 (2005) 473-479.

2.88 fix lb/momentum command

2.88.1 Syntax

```
fix ID group-ID lb/momentum nevery keyword values ...
```

- ID, group-ID are documented in the [fix](#) command
- lb/momentum = style name of this fix command
- nevery = adjust the momentum every this many timesteps
- zero or more keyword/value pairs may be appended
- keyword = *linear*

linear values = xflag yflag zflag
xflag,yflag,zflag = 0/1 to exclude/include each dimension.

2.88.2 Examples

```
fix 1 sphere lb/momentum
fix 1 all lb/momentum linear 1 1 0
```

2.88.3 Description

This fix is based on the [fix momentum](#) command, and was created to be used in place of that command, when a lattice-Boltzmann fluid is present.

Zero the total linear momentum of the system, including both the atoms specified by group-ID and the lattice-Boltzmann fluid every nevery timesteps. If there are no atoms specified by group-ID only the fluid momentum is affected. This is accomplished by adjusting the particle velocities and the fluid velocities at each lattice site.

Note

This fix only considers the linear momentum of the system.

By default, the subtraction is performed for each dimension. This can be changed by specifying the keyword *linear*, along with a set of three flags set to 0/1 in order to exclude/ include the corresponding dimension.

2.88.4 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](#).

2.88.5 Restrictions

Can only be used if a lattice-Boltzmann fluid has been created via the [fix lb/fluid](#) command, and must come after this command.

This fix is part of the LATBOLTZ package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

2.88.6 Related commands

[fix momentum](#), [fix lb/fluid](#)

2.88.7 Default

Zeros the total system linear momentum in each dimension.

2.89 fix lb/viscous command

2.89.1 Syntax

```
fix ID group-ID lb/viscous
```

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

2.89.2 Examples

```
fix 1 flow lb/viscous
```

2.89.3 Description

This fix is similar to the [fix viscous](#) command, and is to be used in place of that command when a lattice-Boltzmann fluid is present using the [fix lb/fluid](#). This should be used in conjunction with one of the built-in LAMMPS integrators, such as [fix NVE](#) or [fix rigid](#).

This fix adds a viscous force to each atom to cause it move with the same velocity as the fluid (an equal and opposite force is applied to the fluid via [fix lb/fluid](#)). When [fix lb/fluid](#) is called with the noise option, the atoms will also experience random forces which will thermalize them to the same temperature as the fluid. In this way, the combination of this fix with [fix lb/fluid](#) and a LAMMPS integrator like [fix NVE](#) is analogous to [fix langevin](#) except here the fluid is explicit. The temperature of the particles can be monitored via the scalar output of [fix lb/fluid](#).

For details of this fix, as well as descriptions and results of several test runs, see [Denniston et al.](#). Please include a citation to this paper if this fix is used in work contributing to published research.

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

As described in the [fix viscous](#) documentation:

“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.”

2.89.5 Restrictions

This fix is part of the LATBOLTZ package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

Can only be used if a lattice-Boltzmann fluid has been created via the [*fix lb/fluid*](#) command, and must come after this command.

2.89.6 Related commands

[*fix lb/fluid*](#)

2.89.7 Default

none

(Denniston et al.) Denniston, C., Afrasiabian, N., Cole-Andre, M.G., Mackay, F. E., Ollila, S.T.T., and Whitehead, T., LAMMPS lb/fluid fix version 2: Improved Hydrodynamic Forces Implemented into LAMMPS through a lattice-Boltzmann fluid, Computer Physics Communications 275 (2022) [108318](#) .

2.90 fix lineforce command

2.90.1 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

2.90.2 Examples

```
fix hold boundary lineforce 0.0 1.0 1.0
```

2.90.3 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.

2.90.4 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.

2.90.5 Restrictions

none

2.90.6 Related commands

fix planeforce

2.90.7 Default

none

2.91 fix manifoldforce command

2.91.1 Syntax

```
fix ID group-ID manifoldforce manifold manifold-args ...
```

- ID, group-ID are documented in *fix* command
- manifold = name of the manifold
- manifold-args = parameters for the manifold

2.91.2 Examples

```
fix constrain all manifoldforce sphere 5.0
```

2.91.3 Description

This fix subtracts each time step from the force the component along the normal of the specified *manifold*. This can be used in combination with *minimize* to remove overlap between particles while keeping them (roughly) constrained to the given manifold, e.g. to set up a run with *fix nve/manifold/rattle*. I have found that only *hftn* and *quickmin* with a very small time step perform adequately though.

2.91.4 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 invoked during [energy minimization](#).

2.91.5 Restrictions

This fix is part of the MANIFOLD package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

Only use this with *min_style hftn* or *min_style quickmin*. If not, the constraints will not be satisfied very well at all. A warning is generated if the *min_style* is incompatible but no error.

2.91.6 Related commands

fix nve/manifold/rattle, fix nvt/manifold/rattle

2.92 fix mdi/qm command

2.92.1 Syntax

```
fix ID group-ID mdi/qm keyword value(s) keyword value(s) ...
```

- ID, group-ID are documented in [fix](#) command
- mdi/qm = style name of this fix command
- zero or more keyword/value pairs may be appended
- keyword = *virial* or *add* or *every* or *connect* or *elements* or *mc*

virial args = yes or no

yes = request virial tensor from server code

no = do not request virial tensor from server code

add args = yes or no

yes = add returned value from server code to LAMMPS quantities

no = do not add returned values to LAMMPS quantities

every args = Nevery

Nevery = request values from server code once every Nevery steps

connect args = yes or no

yes = perform a one-time connection to the MDI engine code

no = do not perform the connection operation

elements args = N_1 N_2 ... N_ntypes

N_1,N_2,...N_ntypes = chemical symbol for each of ntypes LAMMPS atom types

mc args = mcfixedID

mcfixedID = ID of a Monte Carlo fix designed to work with this fix

2.92.2 Examples

```
fix 1 all mdi/qm
fix 1 all mdi/qm virial yes
fix 1 all mdi/qm add no every 100 elements C C H O
```

2.92.3 Description

Added in version 3Aug2022.

This command enables LAMMPS to act as a client with another server code that will compute the total energy, per-atom forces, and total virial for atom conformations and simulation box size/shapes that LAMMPS sends it.

Typically the server code will be a quantum mechanics (QM) code, hence the name of the fix. However this is not required, the server code could be another classical molecular dynamics code or LAMMPS itself. The server code must support use of the [MDI Library](#) as explained below.

Typically, to use this fix, the input script should not define any other classical force field components, e.g. a pair style, bond style, etc.

These are example use cases for this fix, discussed further below:

- perform an ab initio MD (AIMD) simulation with quantum forces
- perform an energy minimization with quantum forces
- perform a nudged elastic band (NEB) calculation with quantum forces
- perform a QM calculation for a series of independent systems which LAMMPS reads or generates once
- run a classical MD simulation and calculate QM energy/forces once every N steps on the current configuration

More generally any command which calculates per-atom forces can instead use quantum forces by defining this fix. Examples are the Monte Carlo commands [fix gcmc](#) and [fix atom/swap](#), as well as the [compute born/matrix](#) command. The only requirement is that internally the command invokes the `post_force()` method of fixes such as this one, which will trigger the quantum calculation.

The code coupling performed by this command is done via the [MDI Library](#). LAMMPS runs as an MDI driver (client), and sends MDI commands to an external MDI engine code (server), e.g. a QM code which has support for MDI. See the [Howto mdi](#) page for more information about how LAMMPS can operate as either an MDI driver or engine.

The examples/mdi directory contains input scripts using this fix in the various use cases discussed below. In each case, two instances of LAMMPS are used, once as an MDI driver, once as an MDI engine (surrogate for a QM code). The examples/mdi/README file explains how to launch two codes so that they communicate via the MDI library using either MPI or sockets. Any QM code that supports MDI could be used in place of LAMMPS acting as a QM surrogate. See the [Howto mdi](#) page for a current list (March 2022) of such QM codes. The examples/QUANTUM directory has examples for coupling LAMMPS to 3 QM codes either via this fix or the [fix mdi/qmmm](#) command.

Note that an engine code can support MDI in either or both of two modes. It can be used as a stand-alone code, launched at the same time as LAMMPS. Or it can be used as a plugin library, which LAMMPS loads. See the [mdi plugin](#) command for how to trigger LAMMPS to load a plugin library. The examples/mdi/README file and examples/QUANTUM/QM-code/README files explain how to launch the two codes in either mode.

The *virial* keyword setting of yes or no determines whether LAMMPS will request the QM code to also compute and return the QM contribution to a stress tensor for the system which LAMMPS will convert to a 6-element symmetric virial tensor.

The *add* keyword setting of *yes* or *no* determines whether the energy and forces and virial returned by the QM code will be added to the LAMMPS internal energy and forces and virial or not. If the setting is *no* then the default [fix_modify energy](#) and [fix_modify virial](#) settings are also set to *no* and your input scripts should not set them to yes. See more details on these fix_modify settings below.

Whatever the setting for the *add* keyword, the QM energy, forces, and virial will be stored by the fix, so they can be accessed by other commands. See details below.

The *every* keyword determines how often the QM code will be invoked during a dynamics run with the current LAMMPS simulation box and configuration of atoms. The QM code will be called once every *Nevery* timesteps. By default *Nevery* = 1.

The *connect* keyword determines whether this fix performs a one-time connection to the QM code. The default is *yes*. The only time a *no* is needed is if this command is used multiple times in an input script and the MDI coupling is between two stand-alone codes (not plugin mode). E.g. if it used inside a loop which also uses the [clear](#) command to destroy the system (including this fix). See the examples/mdi/in.series.driver script as an example of this, where LAMMPS is using the QM code to compute energy and forces for a series of system configurations. In this use case *connect no* is used along with the [mdi connect and exit](#) command to one-time initiate/terminate the connection outside the loop.

The *elements* keyword allows specification of what element each LAMMPS atom type corresponds to. This is specified by the chemical symbol of the element, e.g. C or Al or Si. A symbol must be specified for each of the ntypes LAMMPS atom types. Multiple LAMMPS types can represent the same element. Ntypes is typically specified via the [create_box](#) command or in the data file read by the [read_data](#) command.

If this keyword is specified, then this fix will send the MDI ">ELEMENTS" command to the engine, to ensure the two codes are consistent in their definition of atomic species. If this keyword is not specified, then this fix will send the MDI >TYPES command to the engine. This is fine if both the LAMMPS driver and the MDI engine are initialized so that the atom type values are consistent in both codes.

The *mc* keyword enables this fix to be used with a Monte Carlo (MC) fix to calculate before/after quantum energies as part of the MC accept/reject criterion. The [fix gcmc](#) and [fix atom/swap](#) commands can be used in this manner. Specify the ID of the MC fix following the *mc* keyword. This allows the two fixes to coordinate when MC events are being calculated versus MD timesteps between the MC events.

The following 3 example use cases are illustrated in the examples/mdi directory. See its README file for more details.

(1) To run an ab initio MD (AIMD) dynamics simulation, or an energy minimization with QM forces, or a multi-replica NEB calculation, use *add yes* and *every 1* (the defaults). This is so that every time LAMMPS needs energy and forces, the QM code will be invoked.

Both LAMMPS and the QM code should define the same system (simulation box, atoms and their types) in their respective input scripts. Note that on this scenario, it may not be necessary for LAMMPS to define a pair style or use a neighbor list.

LAMMPS will then perform the timestepping or minimization iterations for the simulation. At the point in each timestep or iteration when LAMMPS needs the force on each atom, it communicates with the engine code. It sends the current simulation box size and shape (if they change dynamically, e.g. during an NPT simulation), and the current atom coordinates. The engine code computes quantum forces on each atom and the total energy of the system and returns them to LAMMPS.

Note that if the AIMD simulation is an NPT or NPH model, or the energy minimization includes [fix box relax](#) to equilibrate the box size/shape, then LAMMPS computes a pressure. This means the *virial* keyword should be set to *yes* so that the QM contribution to the pressure can be included.

(2) To run dynamics with a LAMMPS interatomic potential, and evaluate the QM energy and forces once every 1000 steps, use *add no* and *every 1000*. This could be useful for using an MD run to generate randomized configurations

which are then passed to the QM code to produce training data for a machine learning potential. A `dump custom` command could be invoked every 1000 steps to dump the atom coordinates and QM forces to a file. Likewise the QM energy and virial could be output with the `thermo_style custom` command.

(3) To do a QM evaluation of energy and forces for a series of N independent systems (simulation box and atoms), use `add no` and `every 1`. Write a LAMMPS input script which loops over the N systems. See the [Howto multiple](#) doc page for details on looping and removing old systems. The series of systems could be initialized by reading them from data files with `read_data` commands. Or, for example, by using the `lattice`, `create_atoms`, `delete_atoms`, and/or `displace_atoms random` commands to generate a series of different systems. At the end of the loop perform `run 0` and `write_dump` commands to invoke the QM code and output the QM energy and forces. As in (2) this be useful to produce QM data for training a machine learning potential.

2.92.4 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 computed by the QM code to the global potential energy of the system as part of `thermodynamic output`. The default setting for this fix is `fix_modify energy yes`, unless the `add` keyword is set to `no`, in which case the default setting is `no`.

The `fix_modify virial` option is supported by this fix to add the contribution computed by the QM code to the global pressure of the system as part of `thermodynamic output`. The default setting for this fix is `fix_modify virial yes`, unless the `add` keyword is set to `no`, in which case the default setting is `no`.

This fix computes a global scalar which can be accessed by various `output commands`. The scalar is the energy returned by the QM code. The scalar value calculated by this fix is “extensive”.

This fix also computes a global vector with of length 6 which contains the symmetric virial tensor values returned by the QM code. It can likewise be accessed by various `output commands`.

The ordering of values in the symmetric virial tensor is as follows: vxx, vyy, vzz, vxy, vxz, vyz. The values will be in pressure `units`.

This fix also computes a peratom array with 3 columns which contains the peratom forces returned by the QM code. It can likewise be accessed by various `output commands`.

No parameter of this fix can be used with the `start/stop` keywords of the `run` command.

Assuming the `add` keyword is set to `yes` (the default), the forces computed by the QM code are used during an energy minimization, invoked by the `minimize` command.

Note

If you want the potential energy associated with the QM forces to be included in the total potential energy of the system (the quantity being minimized), you **MUST** not disable the `fix_modify energy` option for this fix, which means the `add` keyword should also be set to `yes` (the default).

2.92.5 Restrictions

This fix is part of the MDI package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

To use LAMMPS as an MDI driver in conjunction with other MDI-enabled codes (MD or QM codes), the [units](#) command should be used to specify *real* or *metal* units. This will ensure the correct unit conversions between LAMMPS and MDI units. The other code will also perform similar unit conversions into its preferred units.

LAMMPS can also be used as an MDI driver in other unit choices it supports, e.g. *lj*, but then no unit conversion to MDI units is performed.

If this fix is used in conjunction with a QM code that does not support periodic boundary conditions (more specifically, a QM code that does not support the `>CELL` MDI command), the LAMMPS system must be fully non-periodic. I.e. no dimension of the system can be periodic.

2.92.6 Related commands

mdi plugin, *mdi engine*, *fix mdi/qmmm*

2.92.7 Default

The default for the optional keywords are *virial* = no, *add* = yes, *every* = 1, *connect* = yes.

2.93 fix mdi/qmmm command

2.93.1 Syntax

```
fix ID group-ID mdi/qmmm mode keyword value(s) keyword value(s) ...
```

- ID, group-ID are documented in [fix](#) command
- mdi/qmmm = style name of this fix command
- mode = *direct* or *potential*
- zero or more keyword/value pairs may be appended
- keyword = *virial* or *add* or *every* or *connect* or *elements*

virial args = yes or no

yes = request virial tensor from server code

no = do not request virial tensor from server code

connect args = yes or no

yes = perform a one-time connection to the MDI engine code

no = do not perform the connection operation

elements args = N_1 N_2 ... N_ntypes

N_1,N_2,...N_ntypes = chemical symbol for each of ntypes LAMMPS atom types

2.93.2 Examples

```
fix 1 all mdi/qmmm direct
fix 1 all mdi/qmmm potential virial yes
fix 1 all mdi/qmmm potential virial yes elements 13 29
```

2.93.3 Description

Added in version 28Mar2023.

This command enables LAMMPS to act as a client with another server code to perform a coupled QM/MM (quantum-mechanics/molecular-mechanics) simulation. LAMMPS will perform classical MD (molecular mechanics or MM) for the (typically larger) MM portion of the system. A quantum mechanics code will calculate quantum energy and forces for the QM portion of the system. The two codes work together to calculate the energy and forces due to the cross interactions between QM and MM atoms. The QM server code must support use of the [MDI Library](#) as explained below.

The partitioning of the system between QM and MM atoms is as follows. Atoms in the specified group are QM atoms; the remaining atoms are MM atoms. The input script should thus define this partitioning. See additional information below about other requirements for an input script to use this fix and perform a QM/MM simulation.

The code coupling performed by this command is done via the [MDI Library](#). LAMMPS runs as an MDI driver (client), and sends MDI commands to an external MDI engine code (server), in this case a QM code which has support for MDI. See the [Howto mdi](#) page for more information about how LAMMPS can operate as either an MDI driver or engine.

The examples/QUANTUM directory has sub-directories with example input scripts using this fix in tandem with different QM codes. The README files in the sub-directories explain how to download and build the various QM codes. They also explain how to launch LAMMPS and the QM code so that they communicate via the MDI library using either MPI or sockets. Any QM code that supports MDI could be used in addition to those discussed in the sub-directories. See the [Howto mdi](#) page for a current list (March 2022) of such QM codes.

Note that an engine code can support MDI in either or both of two modes. It can be used as a stand-alone code, launched at the same time as LAMMPS. Or it can be used as a plugin library, which LAMMPS loads. See the [mdi plugin](#) command for how to trigger LAMMPS to load a plugin library. The examples/QUANTUM sub-directory README files explains how to launch the two codes in either mode.

The *mode* setting determines which QM/MM coupling algorithm is used. LAMMPS currently supports *direct* and *potential* algorithms, based on the *mode* setting. Both algorithms should give reasonably accurate results, but some QM codes support only one of the two modes. E.g. in the examples/QUANTUM directory, PySCF supports only *direct*, NWChem supports only *potential*, and LATTE currently supports neither, so it cannot be used for QM/MM simulations using this fix.

The *direct* option passes the coordinates and charges of each MM atom to the quantum code, in addition to the coordinates of each QM atom. The quantum code returns forces on each QM atom as well as forces on each MM atom. The latter is effectively the force on MM atoms due to the QM atoms.

The input script for performing a *direct* mode QM/MM simulation should do the following:

- delete all bonds (angles, dihedrals, etc) between QM atoms
- set the charge on each QM atom to zero
- define no bonds (angles, dihedrals, etc) which involve both QM and MM atoms
- define a force field (pair, bonds, angles, optional kspace) for the entire system

The first two bullet can be performed using the [delete_bonds](#) and [set](#) commands.

The third bullet is required to have a consistent model, but is not checked by LAMMPS.

The fourth bullet implies that non-bonded non-Coulombic interactions (e.g. van der Waals) between QM/QM and QM/MM pairs of atoms are computed by LAMMPS.

See the examples/QUANTUM/PySCF/in.* files for examples of input scripts for QM/MM simulations using the *direct* mode.

The *potential* option passes the coordinates of each QM atom and a Coulomb potential for each QM atom to the quantum code. The latter is calculated by performing a Coulombics-only calculation for the entire system, subtracting all QM/QM pairwise Coulombic terms, and dividing the Coulomb energy on each QM atom by the charge of the QM atom. The potential value represents the Coulombic influence of all the MM atoms on each QM atom.

The quantum code returns forces and charge on each QM atom. The new charges on the QM atom are used to re-calculate the MM force field, resulting in altered forces on the MM atoms.

The input script for performing a *potential* mode QM/MM simulation should do the following:

- delete all bonds (angles, dihedrals, etc) between QM atoms
- define a hybrid pair style which includes a Coulomb-only pair sub-style
- define no bonds (angles, dihedrals, etc) which involve both QM and MM atoms
- define a force field (pair, bonds, angles, optional kspace) for the entire system

The first operation can be performed using the [delete_bonds](#) command. See the examples/QUANTUM/NWChem/in.* files for examples of how to do this.

The second operation is necessary so that this fix can calculate the Coulomb potential for the QM atoms.

The third bullet is required to have a consistent model, but is not checked by LAMMPS.

The fourth bullet implies that non-bonded non-Coulombic interactions (e.g. van der Waals) between QM/QM and QM/MM pairs of atoms are computed by LAMMPS. However, some QM codes do not want the MM code (LAMMPS) to compute QM/QM van der Waals interactions. NWChem is an example. In this case, the coefficients for those interactions need to be turned off, which typically requires the atom types for the QM atoms be different than those for the MM atoms.

See the examples/QUANTUM/NWChem/in.* files for examples of input scripts for QM/MM simulations using the *potential* mode. Those scripts also illustrate how to turn off QM/QM van der Waals interactions.

The *virial* keyword setting of yes or no determines whether LAMMPS will request the QM code to also compute and return the QM contribution to a stress tensor for the system which LAMMPS will convert to a 6-element symmetric virial tensor.

The *connect* keyword determines whether this fix performs a one-time connection to the QM code. The default is *yes*. The only time a *no* is needed is if this command is used multiple times in an input script. E.g. if it used inside a loop which also uses the [clear](#) command to destroy the system (including this fix). An example would be a script which loops over a series of independent QM/MM simulations, e.g. each with their own data file. In this use case *connect no* could be used along with the [mdi connect and exit](#) command to one-time initiate/terminate the connection outside the loop.

The *elements* keyword allows specification of what element each LAMMPS atom type corresponds to. This is specified by the chemical symbol of the element, e.g. C or Al or Si. A symbol must be specified for each of the ntypes LAMMPS atom types. Multiple LAMMPS types can represent the same element. Ntypes is typically specified via the [create_box](#) command or in the data file read by the [read_data](#) command.

If this keyword is specified, then this fix will send the MDI ">ELEMENTS" command to the engine, to insure the two codes are consistent in their definition of atomic species. If this keyword is not specified, then this fix will send the

MDI >TYPES command to the engine. This is fine if both the LAMMPS driver and the MDI engine are initialized so that the atom type values are consistent in both codes.

2.93.4 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 computed by the QM code to the global potential energy of the system as part of *thermodynamic output*. The default setting for this fix is *fix_modify energy yes*.

The *fix_modify virial* option is supported by this fix to add the contribution computed by the QM code to the global pressure of the system as part of *thermodynamic output*. The default setting for this fix is *fix_modify virial yes*.

This fix computes a global scalar which can be accessed by various *output commands*. The scalar is the energy returned by the QM code. The scalar value calculated by this fix is “extensive”.

This fix also computes a global vector with of length 6 which contains the symmetric virial tensor values returned by the QM code. It can likewise be accessed by various *output commands*.

The ordering of values in the symmetric virial tensor is as follows: vxx, vyy, vzz, vxy, vxz, vyz. The values will be in pressure *units*.

This fix also computes a peratom array with 3 columns which contains the peratom forces returned by the QM code. It can likewise be accessed by various *output commands*. Note that for *direct* mode this will be quantum forces on both QM and MM atoms. For *potential* mode it will only be quantum forces on QM atoms; the forces for MM atoms will be zero.

No parameter of this fix can be used with the *start/stop* keywords of the *run* command.

The forces computed by the QM code are used during an energy minimization, invoked by the *minimize* command.

Note

If you want the potential energy associated with the QM forces to be included in the total potential energy of the system (the quantity being minimized), you MUST not disable the *fix_modify energy* option for this fix.

2.93.5 Restrictions

This command is part of the MDI package. It is only enabled if LAMMPS was built with that package. See the *Build package* page for more info.

To use LAMMPS as an MDI driver in conjunction with other MDI-enabled codes (MD or QM codes), the *units* command should be used to specify *real* or *metal* units. This will ensure the correct unit conversions between LAMMPS and MDI units. The other code will also perform similar unit conversions into its preferred units.

If this fix is used in conjunction with a QM code that does not support periodic boundary conditions (more specifically, a QM code that does not support the >CELL MDI command), the LAMMPS system must be fully non-periodic. I.e. no dimension of the system can be periodic.

2.93.6 Related commands

mdi plugin, mdi engine, fix mdi/qm

2.93.7 Default

The default for the optional keywords are virial = no and connect = yes.

2.94 fix meso/move command

2.94.1 Syntax

```
fix ID group-ID meso/move style args keyword values ...
```

- ID, group-ID are documented in [fix](#) command
- meso/move = style name of this fix command
- style = *linear* or *wiggle* or *rotate* or *variable*

linear args = Vx Vy Vz

Vx,Vy,Vz = components of velocity vector (velocity units), any component can be specified as [NULL](#)

wiggle args = Ax Ay Az period

Ax,Ay,Az = components of amplitude vector (distance units), any component can be specified as [NULL](#)

period = period of oscillation (time units)

rotate args = Px Py Pz Rx Ry Rz period

Px,Py,Pz = origin point of axis of rotation (distance units)

Rx,Ry,Rz = axis of rotation vector

period = period of rotation (time units)

variable args = v_dx v_dy v_dz v_vx v_vy v_vz

v_dx,v_dy,v_dz = 3 variable names that calculate x,y,z displacement as function of time, any component can be specified as [NULL](#)

v_vx,v_vy,v_vz = 3 variable names that calculate x,y,z velocity as function of time, any component can be specified as [NULL](#)

- zero or more keyword/value pairs may be appended

- keyword = *units*

units value = box or lattice

2.94.2 Examples

```
fix 1 boundary meso/move wiggle 3.0 0.0 0.0 1.0 units box
fix 2 boundary meso/move rotate 0.0 0.0 0.0 0.0 0.0 1.0 5.0
fix 2 boundary meso/move variable v_myx v_myy NULL v_VX v_VY NULL
```

2.94.3 Description

Perform updates of position, velocity, internal energy and local density for mesoscopic particles in the group each timestep using the specified settings or formulas, without regard to forces on the particles. This can be useful for boundary, solid bodies or other particles, whose movement can influence nearby particles.

The operation of this fix is exactly like that described by the [fix move](#) command, except that particles' density, internal energy and extrapolated velocity are also updated.

Note

The particles affected by this fix should not be time integrated by other fixes (e.g. [fix sph](#), [fix sph/stationary](#)), since that will change their positions and velocities twice.

Note

As particles move due to this fix, they will pass through 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 sph](#) command). It is up to you to decide whether periodic boundaries are appropriate with the kind of particle motion you are prescribing with this fix.

Note

As discussed below, particles 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 particle. 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 particle. You can reset the image flags (e.g. to 0) before invoking this fix by using the [set image](#) command.

The *linear* style moves particles at a constant velocity, so that their position $X = (x,y,z)$ as a function of time is given in vector notation as

$$X(t) = X_0 + V * \text{delta}$$

where $X_0 = (x_0,y_0,z_0)$ is their position at the time the fix is specified, V is the specified velocity vector with components (V_x,V_y,V_z) , and delta is the time elapsed since the fix was specified. This style also sets the velocity of each particle to $V = (V_x,V_y,V_z)$. 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 sph](#) command would perform, using the corresponding force component on the particle.

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 particles in an oscillatory fashion, so that their position $X = (x,y,z)$ as a function of time is given in vector notation as

$$X(t) = X_0 + A \sin(\omega * \text{delta})$$

where $X0 = (x0,y0,z0)$ is their position at the time the fix is specified, A is the specified amplitude vector with components (A_x, A_y, A_z), ω is $2 \pi / period$, and δt is the time elapsed since the fix was specified. This style also sets the velocity of each particle 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 sph](#) command would perform, using the corresponding force component on the particle.

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 particles around a rotation axis $R = (Rx,Ry,Rz)$ that goes through a point $P = (Px,Py,Pz)$. The *period* of the rotation is also specified. The direction of rotation for the particles 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.

This style also sets the velocity of each particle to $(\omega \times R_{perp})$ where ω is its angular velocity around the rotation axis and R_{perp} is a perpendicular vector from the rotation axis to the particle.

The *variable* style allows the position and velocity components of each particle 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 particle, that can be a function per-atom quantities, such as the particle's position, as well as of the timestep and other simulation values. Note that this fix stores the original coordinates of each particle (see note below) so that per-atom quantity can be used in an atom-style variable formula. See the [variable](#) command for details.

The first 3 variables (`v_dx,v_dy,v_dz`) specified for the *variable* style are used to calculate a displacement from the particle'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 particle.

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 sph](#) command would perform, using the corresponding force component on the particle. 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 particle, 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 particle, and the position of the particle 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/fs 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.

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

This fix writes the original coordinates of moving particles 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.

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 particle. 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](#).

2.94.5 Restrictions

This fix is part of the DPD-SMOOTH package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

This fix requires that atoms store density and internal energy as defined by the [atom_style sph](#) command.

All particles in the group must be mesoscopic SPH/SDPD particles.

Changed in version 29Aug2024.

This fix is incompatible with deformation controls that remap velocity, for instance the *remap v* option of [fix deform](#).

2.94.6 Related commands

[fix move](#), [fix sph](#), [displace_atoms](#)

2.94.7 Default

The option default is units = lattice.

2.95 fix_modify AtC commands

2.95.1 fix_modify AtC add_molecule command

Syntax

```
fix_modify <AtC fixID> add_molecule <small|large> <tag> <group-ID>
```

- AtC fixID = ID of [fix atc](#) instance

- `add_molecule` = name of the AtC sub-command
- `small` or `large` = can be `small` if molecule size < cutoff radius, must be `large` otherwise
- `tag` = tag for tracking a molecule
- `group-ID` = LAMMPS defined group-ID

Examples

```
group WATERGROUP type 1 2
fix_modify AtC add_molecule small water WATERGROUP
```

Description

Associates a tag with all molecules corresponding to a specified group.

Restrictions

None.

Related AtC commands

- [fix_modify AtC command overview](#)
- [fix_modify AtC add_species](#)
- [fix_modify AtC remove_species](#)
- [fix_modify AtC remove_molecule](#)

Default

None.

2.95.2 fix_modify AtC add_species command

Syntax

```
fix_modify <AtC fixID> add_species <tag> <group|type> <ID>
```

- AtC fixID = ID of [fix atc](#) instance
- `add_species` = name of the AtC sub-command
- `tag` = tag for tracking a species
- `group` or `type` = LAMMPS defined group or type of atoms
- `ID` = name of group or type number

Examples

```
fix_modify AtC add_species gold type 1
group GOLDGROUP type 1
fix_modify AtC add_species gold group GOLDGROUP
```

Description

Associates a tag with all atoms of a specified type or within a specified group.

Restrictions

None.

Related AtC commands

- [fix_modify AtC command overview](#)
- [fix_modify AtC add_molecule](#)
- [fix_modify AtC remove_species](#)
- [fix_modify AtC remove_molecule](#)

Default

None.

2.95.3 fix_modify AtC atom_element_map command

Syntax

```
fix_modify <AtC fixID> atom_element_map <eulerian|lagrangian> [<frequency>]
```

- AtC fixID = ID of [fix atc](#) instance
- atom_element_map = name of the AtC sub-command
- *eulerian* or *lagrangian* = frame of reference
- frequency = frequency of updating atom-to-continuum maps based on the current configuration - (only for eulerian)

Examples

```
fix_modify AtC atom_element_map eulerian 100
```

Description

Changes frame of reference from *eulerian* to *lagrangian* or vice versa and sets the frequency for which the map from atoms to elements is reformed and all the attendant data is recalculated.

Restrictions

Cannot change map type after initialization.

Related AtC commands

- *fix_modify AtC command overview*

Default

lagrangian

2.95.4 fix_modify AtC atom_weight command

Syntax

```
fix_modify <AtC fixID> atom_weight <method> <args>
```

- AtC fixID = ID of *fix atc* instance
- atom_weight = name of the AtC sub-command
- method = *constant* or *lattice* or *element* or *region* or *group* or *read_in*
 - *constant* <group-ID> <value>: atoms in specified group are assigned the constant value given
 - *lattice*: volume per atom for specified lattice type (e.g. fcc) and parameter
 - *element*: element volume divided among atoms within element
 - *region*: volume per atom determined based on the atom count in the MD regions and their volumes. Note: meaningful only if atoms completely fill all the regions.
 - *group*: volume per atom determined based on the atom count in a group and its volume
 - *node*: (undocumented)
 - *node_element*: (undocumented)
 - *read_in*<filename>: list of values for atoms are read-in from specified file

Examples

```
fix_modify AtC atom_weight constant myatoms 11.8
fix_modify AtC atom_weight lattice
fix_modify AtC atom_weight read-in atm_wt_file.txt
```

Description

Command for assigning the value of atomic weights used for atomic integration in atom-continuum coupled simulations.

Restrictions

The use of the lattice option requires a lattice type and parameter is already specified.

Related AtC commands

- *fix_modify AtC command overview*

Default

lattice

2.95.5 fix_modify AtC atomic_charge command

Syntax

```
fix_modify <AtC fixID> <include|omit> atomic_charge
```

- AtC fixID = ID of *fix atc* instance
- *include* or *omit* = switch to activate/deactivate inclusion of intrinsic atomic charge in ATC
- atomic_charge = name of the AtC sub-command

Examples

```
fix_modify AtC include atomic_charge
```

Description

Determines whether AtC tracks the total charge as a finite element field.

Restrictions

Required for: *electrostatics*

Related AtC commands

- *fix_modify AtC command overview*

Default

If the atom charge is defined, default is on, otherwise default is off.

2.95.6 fix_modify AtC boundary_dynamics command

Syntax

```
fix_modify <AtC fixID> boundary_dynamics <on|damped_harmonic|prescribed|coupled|none>
```

- AtC fixID = ID of *fix atc* instance
- boundary_dynamics = name of the AtC sub-command
- *on* or *damped_harmonic* *prescribed* *coupled* *none*

Description

Sets different schemes for controlling boundary atoms. *on* will integrate the boundary atoms using the velocity-verlet algorithm. *damped_harmonic* uses a mass/spring/dashpot for the boundary atoms with added arguments of the damping and spring constants followed by the ratio of the boundary type mass to the desired mass. *prescribed* forces the boundary atoms to follow the finite element displacement. *coupled* does the same.

Restrictions

Boundary atoms must be specified. When using swaps between internal and boundary atoms, the initial configuration must have already correctly partitioned the two.

Related AtC commands

- *fix_modify AtC command overview*

Default

prescribed on

2.95.7 fix_modify AtC boundary_faceset command

Syntax

```
fix_modify <AtC fixID> boundary_faceset <is|add> <faceset_name>
```

- AtC fixID = ID of *fix atc* instance
- boundary_faceset = name of the AtC sub-command
- *is* or *add* = select whether to select or add a faceset
- faceset_name = name of the faceset

Examples

```
fix_modify AtC boundary_faceset is obndy
```

Description

This command species the faceset name when using a faceset to compute the MD/FE boundary fluxes. The faceset must already exist.

Restrictions

This is only valid when *fe_md_boundary* is set to *faceset*.

Related AtC commands

- *fix_modify AtC command overview*
- *fix_modify AtC fe_md_boundary*
- *fix_modify AtC mesh create_faceset box*
- *fix_modify AtC mesh create_faceset plane*

Default

None.

2.95.8 fix_modify AtC boundary type command

Syntax

```
fix_modify <AtC fixID> boundary type <atom-type-id>
```

- AtC fixID = ID of *fix atc* instance
- boundary type = name of the AtC sub-command
- atom-type-id = type id for atoms that represent a fictitious boundary internal to the FE mesh

Examples

```
fix_modify AtC boundary type ghost_atoms
```

Description

Command to define the atoms that represent the fictitious boundary internal to the FE mesh. For fully overlapped MD/FE domains with periodic boundary conditions no boundary atoms should be defined.

Restrictions

None.

Related AtC commands

- *fix_modify AtC command overview*

Default

None.

2.95.9 fix_modify AtC consistent_fe_initialization command

Syntax

```
fix_modify <AtC fixID> consistent_fe_initialization <on|off>
```

- AtC fixID = ID of *fix atc* instance
- consistent_fe_initialization = name of the AtC sub-command
- *on* or *off* = switch to activate/deactivate the initial setting of the FE intrinsic field to match the projected MD field

Examples

```
fix_modify AtC consistent_fe_initialization on
```

Description

Determines whether AtC initializes FE intrinsic fields (e.g., temperature) to match the projected MD values. This is particularly useful for fully overlapping simulations.

Restrictions

Can be used with: *thermal*, *two_temperature*. Cannot be used with time filtering on. Does not include boundary nodes.

Related AtC commands

- *fix_modify AtC command overview*

Default

Default is *off*

2.95.10 fix_modify AtC control localized_lambda command

Syntax

```
fix_modify <AtC fixID> control localized_lambda <on|off>
```

- AtC fixID = ID of *fix atc* instance
- control localized_lambda = name of the AtC sub-command
- *on* or *off* = Toggles state of localization algorithm

Examples

```
fix_modify AtC control localized_lambda on
```

Description

Turns the localization algorithms *on* or *off* for control algorithms to restrict the influence of FE coupling or boundary conditions to a region near the boundary of the MD region. Control algorithms will not affect atoms in elements not possessing faces on the boundary of the region. Flux-based control is localized via row-sum lumping while quantity control is done by solving a truncated matrix equation.

Restrictions

None.

Related AtC commands

- *fix_modify AtC command overview*

Default

off

2.95.11 fix_modify AtC control momentum command

Syntax

```
fix_modify <AtC fixID> control <physics_type> <solution_parameter> <value>
fix_modify AtC control momentum none
fix_modify AtC control momentum rescale <frequency>
fix_modify AtC control momentum glc_displacement
fix_modify AtC control momentum glc_velocity
fix_modify AtC control momentum hoover
fix_modify AtC control momentum flux [faceset face_set_id, interpolate]
```

- AtC fixID = ID of *fix atc* instance
- control = name of the AtC sub-command
- physics_type = *thermal* or *momentum*
- solution_parameter = *max_iterations* or *tolerance*
- value = solution_parameter value
- momentum option = *none* or *rescale* or *glc_displacement* or *glc_velocity* *hoover* or *flux*
- frequency = time step frequency for applying displacement and velocity rescaling
- faceset_id = id of boundary face set (optional, only for *faceset*)

Examples

```
fix_modify AtC control momentum none
fix_modify AtC control momentum flux faceset bndy_faces
fix_modify AtC control momentum glc_velocity
```

Description

The general version of *control* sets the numerical parameters for the matrix solvers used in the specified control algorithm. Many solution approaches require iterative solvers, and these methods enable users to provide the maximum number of iterations and the relative tolerance.

The *control momentum* version sets the momentum exchange mechanism from the finite elements to the atoms, managed through a control algorithm. *rescale* computes a scale factor for each atom to match the finite element temperature. *hoover* is a Gaussian least-constraint isokinetic thermostat enforces that the nodal restricted atomic temperature matches the finite element temperature. *flux* is a similar mode, but rather adds energy to the atoms based on conservation of energy.

correction_max_iterations sets the maximum number of iterations to compute the second order in time correction term for lambda with the fractional step method. The method uses the same tolerance as the controller's matrix solver.

Restrictions

Only for be used with the specific controllers *thermal* or *momentum*. They are ignored if a lumped solution is requested. *control momentum* is only for be used with specific transfers: elastic *rescale* not valid with time filtering activated

Related AtC commands

- *fix_modify AtC command overview*
- *fix_modify AtC control thermal*

Default

- *max_iterations* is the number of rows in the matrix.
- *tolerance* is 1.0e-10.

2.95.12 fix_modify AtC control thermal command

Syntax

```
fix_modify <AtC fixID> control <physics_type> <solution_parameter> <value>
fix_modify <AtC fixID> control thermal <control_type> <optional_args>
fix_modify <AtC fixID> control thermal rescale <frequency>
fix_modify <AtC fixID> control thermal flux <boundary_integration_type> <faceset_id>
fix_modify <AtC fixID> control thermal correction_max_iterations <max_iterations>
```

- AtC fixID = ID of *fix atc* instance
- control = name of the AtC sub-command
- physics_type = *thermal* or *momentum*
- solution_parameter = *max_iterations* or *tolerance*
- value = solution_parameter value
- thermal control_type = *none* or *rescale* or *hoover* or *flux*
- frequency = time step frequency for applying velocity rescaling

- `boundary_integration_type` = *faceset* or *interpolate* (optional)
- `faceset_id` = id of boundary face set (optional, only for *faceset*)
- `correction_max_iterations` = maximum number of iterations that will be used by iterative matrix solvers for *thermal* physics type

Examples

```
fix_modify AtC control thermal none
fix_modify AtC control thermal rescale 10
fix_modify AtC control thermal hoover
fix_modify AtC control thermal flux
fix_modify AtC control thermal flux faceset bndy_faces
fix_modify AtC control thermal correction_max_iterations 10
```

Description

The general version of *control* sets the numerical parameters for the matrix solvers used in the specified control algorithm. Many solution approaches require iterative solvers, and these methods enable users to provide the maximum number of iterations and the relative tolerance.

The *control thermal* version sets the energy exchange mechanism from the finite elements to the atoms, managed through a control algorithm. *rescale* computes a scale factor for each atom to match the finite element temperature. *hoover* is a Gaussian least-constraint isokinetic thermostat enforces that the nodal restricted atomic temperature matches the finite element temperature. *flux* is a similar mode, but rather adds energy to the atoms based on conservation of energy. *hoover* and *flux* allow the prescription of sources or fixed temperatures on the atoms.

correction_max_iterations sets the maximum number of iterations to compute the second order in time correction term for lambda with the fractional step method. The method uses the same tolerance as the controller's matrix solver.

Restrictions

Only for be used with the specific controllers *thermal* or *momentum*. They are ignored if a lumped solution is requested. *control thermal* is only for be used with specific transfers: *thermal* (*rescale*, *hoover*, *flux*), *two_temperature* (*flux*). *rescale* not valid with time filtering activated

correction_max_iterations is only for use with *thermal* physics using the fractional step method.

Related AtC commands

- [fix_modify AtC command overview](#)
- [fix_modify AtC control momentum](#)

Default

- *max_iterations* is the number of rows in the matrix.
- *tolerance* is 1.0e-10.
- *rescale* frequency is 1
- *flux boundary_integration_type* is *interpolate*
- *correction_max_iterations* is 20

2.95.13 fix_modify AtC decomposition command

Syntax

```
fix_modify <AtC fixID> decomposition <type>
```

- AtC fixID = ID of *fix atc* instance
- decomposition = name of the AtC sub-command
- type = *replicated_memory* or *distributed_memory*

Examples

```
fix_modify AtC decomposition distributed_memory
```

Description

Command for assigning the distribution of work and memory for parallel runs. With *replicated_memory* the nodal information is replicated on each processor, and with *distributed_memory* only the owned nodal information kept on each processor. The *replicated_memory* option is most appropriate for simulations were the number of nodes is much smaller than the number of atoms.

Restrictions

None.

Related AtC commands

- *fix_modify AtC command overview*

Default

replicated_memory

2.95.14 fix_modify AtC extrinsic electron_integration command

Syntax

```
fix_modify <AtC fixID> extrinsic electron_integration <integration_type> [<num_subcycle_steps>]
```

- AtC fixID = ID of [fix atc](#) instance
- extrinsic electron_integration = name of the AtC sub-command
- integration_type = *explicit* or *implicit* or *steady*
- num_subcycle_steps = number of subcycle steps for the electron time integration (optional)

Examples

```
fix_modify AtC extrinsic electron_integration implicit  
fix_modify AtC extrinsic electron_integration explicit 100
```

Description

Switches between integration schemes for the electron temperature. The number of subcycling steps used to integrate the electron temperature for one LAMMPS timestep can be manually adjusted to capture fast electron dynamics.

Restrictions

For use only with the two_temperature type of the AtC fix (see [fix atc](#) command)

Related AtC commands

- [fix_modify AtC command overview](#)

Default

implicit and *subcycle_steps* = 1

2.95.15 fix_modify AtC equilibrium_start command

Syntax

```
fix_modify <AtC fixID> equilibrium_start <on|off>
```

- AtC fixID = ID of *fix atc* instance
- equilibrium_start = name of the AtC sub-command
- *exponential* or *step* or *no_filter* = select type of filter

Examples

```
fix_modify AtC equilibrium_start on
```

Description

Starts filtered calculations assuming they start in equilibrium, i.e. perfect finite element force balance.

Restrictions

Only for use with these specific transfers: thermal, two_temperature

Related AtC commands

- *fix_modify AtC command overview*
- *fix_modify AtC filter*
- *fix_modify AtC filter scale*

Default

None.

2.95.16 fix_modify AtC extrinsic exchange command

Syntax

```
fix_modify <AtC fixID> extrinsic exchange <on|off>
```

- AtC fixID = ID of *fix atc* instance
- extrinsic exchange = name of the AtC sub-command
- *on* or *off* = set state of energy exchange

Examples

```
fix_modify AtC extrinsic exchange on
```

Description

Switches energy exchange between the MD system and the electron system on or off

Restrictions

For use only with the two_temperature type of the AtC fix (see [fix atc](#) command)

Related AtC commands

- [fix_modify AtC command overview](#)

Default

on

2.95.17 fix_modify AtC fe_md_boundary command

Syntax

```
fix_modify <AtC fixID> fe_md_boundary <faceset|interpolate|no_boundary>
```

- AtC fixID = ID of [fix atc](#) instance
- fe_md_boundary = name of the AtC sub-command
- *faceset* or *interpolate* or *no_boundary*

Examples

```
fix_modify AtC fe_md_boundary interpolate
```

Description

Specifies different methods for computing fluxes between between the MD and FE integration regions. Faceset defines a faceset separating the MD and FE regions and uses finite element face quadrature to compute the flux. Interpolate uses a reconstruction scheme to approximate the flux, which is more robust but less accurate if the MD/FE boundary does correspond to a faceset. No boundary results in no fluxes between the systems being computed.

Restrictions

If *faceset* is used, all the AtC non-boundary atoms must lie within and completely fill the domain enclosed by the faceset.

Related AtC commands

- *fix_modify AtC command overview*
- *fix_modify AtC mesh create_faceset box*
- *fix_modify AtC mesh create_faceset plane*

Default

interpolate

2.95.18 fix_modify AtC filter scale command

Syntax

```
fix_modify <AtC fixID> filter scale <scale>
```

- AtC fixID = ID of *fix atc* instance
- filter scale = name of the AtC sub-command
- scale = characteristic times scale of the filter

Examples

```
fix_modify AtC filter scale 10.0
```

Description

Sets the time scale for MD dynamics filter to construct a more appropriate continuous field.

Restrictions

Only for use with these specific transfers: thermal, two_temperature

Related AtC commands

- [*fix_modify AtC command overview*](#)
- [*fix_modify AtC filter*](#)
- [*fix_modify AtC filter type*](#)

Default

0.0

2.95.19 fix_modify AtC filter type command

Syntax

```
fix_modify <AtC fixID> filter type <exponential|step|no_filter>
```

- AtC fixID = ID of [*fix atc*](#) instance
- filter type = name of the AtC sub-command
- *exponential* or *step* or *no_filter* = select type of filter

Examples

```
fix_modify AtC filter type exponential
```

Description

Specifies the type of time filter used.

Restrictions

Only for use with these specific transfers: thermal, two_temperature

Related AtC commands

- [*fix_modify AtC command overview*](#)
- [*fix_modify AtC filter*](#)
- [*fix_modify AtC filter scale*](#)

Default

None.

2.95.20 fix_modify AtC fix command

Syntax

```
fix_modify <AtC fixID> fix <field> <nodeset> <constant|function>
```

- AtC fixID = ID of *fix atc* instance
- fix = name of the AtC sub-command
- field = field kind name valid for type of physics: *temperature* or *electron_temperature*
- nodeset = name of set of nodes to apply boundary condition
- constant or function = value or name of function followed by its parameters

Examples

```
fix_modify AtC fix temperature groupNAME 10.  
fix_modify AtC fix temperature groupNAME 0 0 0 10.0 0 0 1.0
```

Description

Creates a constraint on the values of the specified field at specified nodes.

Restrictions

The keyword *all* is reserved and thus not available as nodeset name.

Related AtC commands

- *fix_modify AtC command overview*
- *fix_modify AtC unfix*

Default

None.

2.95.21 fix_modify AtC fix_flux command

Syntax

```
fix_modify <AtC fixID> fix_flux <field> <face_set> <value|function>
```

- AtC fixID = ID of [fix atc](#) instance
- fix_flux = name of the AtC sub-command
- field = field kind name valid for type of physics: *temperature* or *electron_temperature*
- face_set = name of set of element faces
- value or function = value or name of function followed by its parameters

Examples

```
fix_modify AtC fix_flux temperature faceSet 10.0
```

Description

Command for fixing normal fluxes e.g. heat_flux. This command only prescribes the normal component of the physical flux, e.g. heat (energy) flux. The units are in AtC units, i.e. derived from the LAMMPS length, time, and mass scales.

Restrictions

Only normal fluxes (Neumann data) can be prescribed.

Related AtC commands

- [fix_modify AtC command overview](#)
- [fix_modify AtC unfix_flux](#)

Default

None.

2.95.22 fix_modify AtC computes command

Syntax

```
fix_modify <AtC fixID> computes <add|delete> <per-atom compute-ID> <volume|number>
```

- AtC fixID = ID of [fix atc](#) instance
- computes = name of the AtC sub-command
- add or delete = add or delete the calculation of an equivalent continuum field for the specified per-atom compute as volume or number density quantity

- per-atom compute-ID = ID of a per-atom compute; fields can be calculated for all per-atom computes available in LAMMPS
- *volume* or *number* = select whether the created field is a per-unit-volume quantity or a per-atom quantity as weighted by kernel functions

Examples

```
compute virial all stress/atom
fix_modify AtC computes add virial volume
fix_modify AtC computes delete virial

compute centrosymmetry all centro/atom
fix_modify AtC computes add centrosymmetry number
```

Description

Calculates continuum fields corresponding to specified per-atom *computes* created by LAMMPS.

Restrictions

Must be used with *fix atc hardy*. The per-atom compute must be specified before the corresponding continuum field can be requested.

Related AtC commands

- *fix_modify AtC command overview*
- *fix_modify AtC fields*
- *compute*

Default

None.

2.95.23 fix_modify AtC fields command

Syntax

```
fix_modify <AtC fixID> fields <all|none>
fix_modify <AtC fixID> fields <add|delete> <list_of_fields>
```

- AtC fixID = ID of *fix atc* instance
- fields = name of the AtC sub-command
- *all* or *none* = output all or no fields
- *add* or *delete* = add or delete the listed output fields
- *list_of_fields* = one or more of the fields listed below:

- density : mass per unit volume
- displacement : displacement vector
- momentum : momentum per unit volume
- velocity : defined by momentum divided by density
- projected_velocity : simple kernel estimation of atomic velocities
- temperature : temperature derived from the relative atomic kinetic energy
- kinetic_temperature : temperature derived from the full kinetic energy
- number_density : simple kernel estimation of number of atoms per unit volume
- stress : Cauchy stress tensor for eulerian analysis (atom_element_map), or first Piola-Kirchhoff stress tensor for lagrangian analysis
- transformed_stress : first Piola-Kirchhoff stress tensor for eulerian analysis (atom_element_map), or Cauchy stress tensor for lagrangian analysis
- heat_flux : spatial heat flux vector for eulerian, or referential heat flux vector for lagrangian
- potential_energy : potential energy per unit volume
- kinetic_energy : kinetic energy per unit volume
- thermal_energy : thermal energy (kinetic energy - continuum kinetic energy) per unit volume
- internal_energy : total internal energy (potential + thermal) per unit volume
- energy : total energy (potential + kinetic) per unit volume
- number_density : number of atoms per unit volume
- eshelby_stress : configurational stress (energy-momentum) tensor defined by (*Eshelby*)
- vacancy_concentration : volume fraction of vacancy content
- type_concentration : volume fraction of a specific atom type

Examples

```
fix_modify AtC fields add velocity temperature
```

Description

Allows modification of the fields calculated and output by the AtC transfer class. The commands are cumulative, e.g.:

```
fix_modify AtC fields none
fix_modify AtC fields add velocity temperature
```

will only output the velocity and temperature fields.

Restrictions

Must be used with [fix atc hardy](#). Currently, the stress and heat flux formulas are only correct for central force potentials, e.g. Lennard-Jones and EAM but not Stillinger-Weber.

Related AtC commands

- [fix_modify AtC command overview](#)
- [fix_modify AtC gradients](#)
- [fix_modify AtC rates](#)
- [fix_modify AtC computes](#)

Default

By default, no fields are output.

References

(Eshelby) J.D. Eshelby, Philos. Trans. Royal Soc. London A, Math. Phys. Sci., Vol. 244, No. 877 (1951) pp. 87-112; J. Elasticity, Vol. 5, Nos. 3-4 (1975) pp. 321-335]

2.95.24 fix_modify AtC gradients command

Syntax

```
fix_modify <AtC fixID> gradients <add|delete> <list_of_fields>
```

- AtC fixID = ID of [fix atc](#) instance
- gradients = name of the AtC sub-command
- *add* or *delete* = select whether to add or delete calculation of gradients for the listed output fields
- list_of_fields = one or more of the fields listed below:
 - density : mass per unit volume
 - displacement : displacement vector
 - momentum : momentum per unit volume
 - velocity : defined by momentum divided by density
 - projected_velocity : simple kernel estimation of atomic velocities
 - temperature : temperature derived from the relative atomic kinetic energy
 - kinetic_temperature : temperature derived from the full kinetic energy
 - number_density : simple kernel estimation of number of atoms per unit volume
 - stress : Cauchy stress tensor for eulerian analysis (atom_element_map), or first Piola-Kirchhoff stress tensor for lagrangian analysis
 - transformed_stress : first Piola-Kirchhoff stress tensor for eulerian analysis (atom_element_map), or Cauchy stress tensor for lagrangian analysis

- heat_flux : spatial heat flux vector for eulerian, or referential heat flux vector for lagrangian
- potential_energy : potential energy per unit volume
- kinetic_energy : kinetic energy per unit volume
- thermal_energy : thermal energy (kinetic energy - continuum kinetic energy) per unit volume
- internal_energy : total internal energy (potential + thermal) per unit volume
- energy : total energy (potential + kinetic) per unit volume
- number_density : number of atoms per unit volume
- eshelby_stress : configurational stress (energy-momentum) tensor defined by (*Eshelby*)
- vacancy_concentration : volume fraction of vacancy content
- type_concentration : volume fraction of a specific atom type

Examples

```
fix_modify AtC gradients add temperature velocity stress
fix_modify AtC gradients delete velocity
```

Description

Requests calculation and output of gradients of the fields from the AtC transfer class. These gradients will be with regard to spatial or material coordinate for Eulerian or Lagrangian analysis, respectively, as specified by [fix_modify AtC atom_element_map](#)

Restrictions

Must be used with [fix atc hardy](#).

Related AtC commands

- [fix_modify AtC command overview](#)
- [fix_modify AtC atom_element_map](#)
- [fix_modify AtC fields](#)
- [fix_modify AtC rates](#)

Default

None.

References

(**Eshelby**) J.D. Eshelby, Philos. Trans. Royal Soc. London A, Math. Phys. Sci., Vol. 244, No. 877 (1951) pp. 87-112; J. Elasticity, Vol. 5, Nos. 3-4 (1975) pp. 321-335]

2.95.25 fix_modify AtC kernel command

Syntax

```
fix_modify <AtC fixID> kernel <type> <parameters>
```

- AtC fixID = ID of [fix atc](#) instance
- kernel = name of the AtC sub-command
- type = *step* or *cell* or *cubic_bar* or *cubic_cylinder* or *cubic_sphere* or *quartic_bar* or *quartic_cylinder* or *quartic_sphere*
- the following parameter(s) are required for each kernel:
 - *step* : <radius>
 - *cell* : <hx> <hy> <hz> or <h>
 - *cubic_bar* : <half_width>
 - *cubic_cylinder* : <radius>
 - *cubic_sphere* : <radius>
 - *quartic_bar* : <half_width>
 - *quartic_cylinder* : <radius>
 - *quartic_sphere* : <radius>

Examples

```
fix_modify AtC kernel cell 1.0 1.0 1.0
fix_modify AtC kernel quartic_sphere 10.0
```

Description

Sets the localization kernel type and parameters for [fix atc hardy](#).

Restrictions

Must be used with [fix atc hardy](#). For bar kernel types, half-width oriented along x-direction. For cylinder kernel types, cylindrical axis is assumed to be in z-direction.

Related AtC commands

- [fix_modify AtC command overview](#)
- [fix_modify AtC fields](#)
- [fix_modify AtC gradients](#)
- [fix_modify AtC rates](#)
- [fix_modify AtC computes](#)

Default

None.

2.95.26 fix_modify AtC on_the_fly command

Syntax

```
fix_modify <AtC fixID> on_the_fly <bond|kernel> <on|off>
```

- AtC fixID = ID of [fix atc](#) instance
- on_the_fly = name of the AtC sub-command
- *bond* or *kernel* = specifies on-the-fly calculation of *bond* or *kernel* matrix elements
- *on* or *off* = activate or discontinue on-the-fly mode

Examples

```
fix_modify AtC on_the_fly bond on
fix_modify AtC on_the_fly kernel
fix_modify AtC on_the_fly kernel off
```

Description

Overrides normal mode of pre-calculating and storing bond pair-to-node and kernel atom-to-node matrices. If activated, it will calculate elements of these matrices during repeated calls of field computations (i.e. “on-the-fly”) and not store them for future use. The *on* flag is optional - if omitted, *on_the_fly* will be activated for the specified matrix. Can be deactivated using the *off* flag.

Restrictions

Must be used with [fix atc hardy](#).

Related AtC commands

- [fix_modify AtC command overview](#)

Default

By default, on-the-fly calculation is not active (i.e. off). However, THE code does a memory allocation check to determine if it can store all needed bond and kernel matrix elements. If this allocation fails, on-the-fly will be activated.

2.95.27 fix_modify AtC rates command

Syntax

```
fix_modify <AtC fixID> rates <add|delete> <list_of_fields>
```

- AtC fixID = ID of [fix atc](#) instance
- rates = name of the AtC sub-command
- *add* or *delete* = select whether to add or delete calculation of rates for the listed output fields
- list_of_fields = one or more of the fields listed below:
 - density : mass per unit volume
 - displacement : displacement vector
 - momentum : momentum per unit volume
 - velocity : defined by momentum divided by density
 - projected_velocity : simple kernel estimation of atomic velocities
 - temperature : temperature derived from the relative atomic kinetic energy
 - kinetic_temperature : temperature derived from the full kinetic energy
 - number_density : simple kernel estimation of number of atoms per unit volume
 - stress : Cauchy stress tensor for eulerian analysis (atom_element_map), or first Piola-Kirchhoff stress tensor for lagrangian analysis
 - transformed_stress : first Piola-Kirchhoff stress tensor for eulerian analysis (atom_element_map), or Cauchy stress tensor for lagrangian analysis
 - heat_flux : spatial heat flux vector for eulerian, or referential heat flux vector for lagrangian
 - potential_energy : potential energy per unit volume
 - kinetic_energy : kinetic energy per unit volume
 - thermal_energy : thermal energy (kinetic energy - continuum kinetic energy) per unit volume
 - internal_energy : total internal energy (potential + thermal) per unit volume
 - energy : total energy (potential + kinetic) per unit volume

- number_density : number of atoms per unit volume
- eshelby_stress : configurational stress (energy-momentum) tensor defined by (*Eshelby*)
- vacancy_concentration : volume fraction of vacancy content
- type_concentration : volume fraction of a specific atom type

Examples

```
fix_modify AtC rates add temperature velocity stress
fix_modify AtC rates delete stress
```

Description

Requests calculation and output of rates (time derivatives) of the fields from the AtC transfer class. For Eulerian analysis (see [fix_modify AtC atom_element_map](#)) these rates are the partial time derivatives of the nodal fields, not the full (material) time derivatives.

Restrictions

Must be used with [fix atc hardy](#).

Related AtC commands

- [fix_modify AtC command overview](#)
- [fix_modify AtC atom_element_map](#)
- [fix_modify AtC fields](#)
- [fix_modify AtC fields](#)

Default

None.

References

(**Eshelby**) J.D. Eshelby, Philos. Trans. Royal Soc. London A, Math. Phys. Sci., Vol. 244, No. 877 (1951) pp. 87-112; J. Elasticity, Vol. 5, Nos. 3-4 (1975) pp. 321-335]

2.95.28 fix_modify AtC initial command

Syntax

```
fix_modify <AtC fixID> initial <field> <nodeset> <constant|function>
```

- AtC fixID = ID of [fix atc](#) instance
- initial = name of the AtC sub-command
- field = field kind name valid for type of physics: *temperature* or *electron_temperature*
- nodeset = name of set of nodes to apply initial condition
- *constant* or *function* = value or name of function followed by its parameters

Examples

```
fix_modify AtC initial temperature groupNAME 10.
```

Description

Sets the initial values for the specified field at the specified nodes.

Restrictions

The keyword *all* is reserved and thus not available as nodeset name.

Related AtC commands

- [fix_modify AtC command overview](#)

Default

None.

2.95.29 fix_modify AtC internal_element_set command

Syntax

```
fix_modify <AtC fixID> internal_element_set <element_set_name>
```

- AtC fixID = ID of [fix atc](#) instance
- internal_element_set = name of the AtC sub-command
- element_set_name = name of element set defining internal region, or *off*

Examples

```
fix_modify AtC internal_element_set myElementSet
fix_modify AtC internal_element_set off
```

Description

Enables AtC to base the region for internal atoms to be an element set. If no ghost atoms are used, all the AtC atoms must be constrained to remain in this element set by the user, e.g., with walls. If boundary atoms are used in conjunction with Eulerian atom maps AtC will partition all atoms of a boundary or internal type to be of type internal if they are in the internal region or to be of type boundary otherwise.

Restrictions

If boundary atoms are used in conjunction with Eulerian atom maps, the Eulerian reset frequency must be an integer multiple of the Lammmps rereighbor frequency.

Related AtC commands

- [fix_modify AtC command overview](#)
- [fix_modify AtC atom_element_map](#)
- [fix_modify AtC boundary type](#)

Default

off

2.95.30 fix_modify AtC internal_quadrature command

Syntax

```
fix_modify <AtC fixID> internal_quadrature <on|off> [region]
```

- AtC fixID = ID of [fix atc](#) instance
- internal_quadrature = name of the AtC sub-command
- on or off = select whether internal quadrature is enabled or not
- region = treat finite elements as within MD region (optional)

Examples

```
fix_modify AtC internal_quadrature off
```

Description

Command to use or not use atomic quadrature on internal elements fully filled with atoms. By turning the internal quadrature off these elements do not contribute to the governing PDE and the fields at the internal nodes follow the weighted averages of the atomic data.

Optional region tag specifies which finite element nodes will be treated as being within the MD region. This option is only valid with internal_quadrature off.

Related AtC commands

- *fix_modify AtC command overview*

Default

on.

2.95.31 fix_modify AtC kernel_bandwidth command

Syntax

```
fix_modify <AtC fixID> kernel_bandwidth <value>
```

- AtC fixID = ID of *fix atc* instance
- kernel_bandwidth = name of the AtC sub-command
- value = new bandwidth value

Examples

```
fix_modify AtC kernel_bandwidth 8
```

Description

Sets a maximum parallel bandwidth for the kernel functions during parallel communication. If the command is not issued, the default will be to assume the bandwidth of the kernel matrix corresponds to the number of sampling locations.

Restrictions

Only is used if kernel functions are being used.

Related AtC commands

- *fix_modify AtC command overview*

Default

Number of sample locations.

2.95.32 fix_modify AtC control lumped_lambda_solve command

Syntax

```
fix_modify <AtC fixID> control lumped_lambda_solve <on|off>
```

- AtC fixID = ID of *fix atc* instance
- control lumped_lambda_solve = name of the AtC sub-command
- *on* or *off* = Toggles state of lumped matrix

Examples

```
fix_modify AtC control lumped_lambda_solve on
```

Description

Command select whether to use or not use lumped matrix for lambda solve.

Restrictions

None.

Related AtC commands

- *fix_modify AtC command overview*

Default

off

2.95.33 fix_modify AtC control mask_direction command**Syntax**

```
fix_modify <AtC fixID> control mask_direction <direction> <on|off>
```

- AtC fixID = ID of *fix atc* instance
- control mask_direction = name of the AtC sub-command
- direction = select direction
- *on* or *off* = Toggles state

Examples

```
fix_modify AtC control mask_direction 0 on
```

Description

Command to mask out certain dimensions from the atomic regulator

Restrictions

None.

Related AtC commands

- *fix_modify AtC command overview*

2.95.34 fix_modify AtC mass_matrix command**Syntax**

```
fix_modify <AtC fixID> mass_matrix <fe|md_fe>
```

- AtC fixID = ID of *fix atc* instance
- mass_matrix = name of the AtC sub-command
- *fe* or *md_fe* = activate/deactivate using the FE mass matrix in the MD region

Examples

```
fix_modify AtC mass_matrix fe
```

Description

Determines whether AtC uses the FE mass matrix based on Gaussian quadrature or based on atomic quadrature in the MD region. This is useful for fully overlapping simulations to improve efficiency.

Restrictions

Should not be used unless the FE region is contained within the MD region, otherwise the method will be unstable and inaccurate.

Related AtC commands

- [fix_modify AtC command overview](#)

Default

md_fe

2.95.35 fix_modify AtC material command

Syntax

```
fix_modify <AtC fixID> material <elementset_name> <material_id>
```

- AtC fixID = ID of [fix atc](#) instance
- material = name of the AtC sub-command
- elementset_name = name of the elementset
- material_id = ID of the material

Examples

```
fix_modify AtC material gap_region 1
```

Description

Sets the material model in *elementset_name* to be of type *material_id*.

Restrictions

The element set must already be created and the material must be specified in the material file given the the atc fix on construction

Related AtC commands

- *fix_modify AtC command overview*

Default

All elements default to the first material in the material file.

2.95.36 fix_modify AtC mesh add_to_nodeset command

Syntax

```
fix_modify <AtC fixID> mesh add_to_nodeset <id> <xmin> <xmax> <ymin> <ymax> <zmin>
    <zmax>
```

- AtC fixID = ID of *fix atc* instance
- mesh create_nodeset = name of the AtC sub-command
- id = id to assign to the collection of FE nodes
- <xmin> <xmax> <ymin> <ymax> <zmin> <zmax> = coordinates of the bounding box that contains the desired nodes to be added

Examples

```
fix_modify AtC mesh add_to_nodeset lbc -12.1 -11.9 -12 12 -12 12
```

Description

Command to add nodes to an already existing FE nodeset.

Restrictions

None

Related AtC commands

- [fix_modify AtC command overview](#)
- [fix_modify AtC mesh create_nodeset](#)

Default

Coordinates are assumed to be in lattice units.

2.95.37 fix_modify AtC mesh create command

Syntax

```
fix_modify <AtC fixID> mesh create <nx> <ny> <nz> <region-ID> <f|p> <f|p> <f|p>
```

- AtC fixID = ID of [fix atc](#) instance
- mesh create = name of the AtC sub-command
- nx ny nz = number of elements in x-, y-, and z-direction
- region-ID = ID of region that is to be meshed
- f or p = periodicity flags for x-, y-, and z-direction

Examples

```
fix_modify AtC mesh create 10 1 1 feRegion p p p
```

Description

Creates a uniform mesh in a rectangular region.

Restrictions

Creates only uniform rectangular grids in a rectangular region

Related AtC commands

- [fix_modify AtC command overview](#)
- [fix_modify AtC mesh quadrature](#)

Default

When created, the mesh defaults to gauss2 (2-point Gaussian) quadrature. Use the [fix_modify AtC mesh quadrature](#) command to change the quadrature style.

2.95.38 fix_modify AtC mesh create_elementset command

Syntax

```
fix_modify <AtC fixID> mesh create_elementset <id> <xmin> <xmax> <ymin> <ymax> <zmin>
           ↵<zmax>
```

- AtC fixID = ID of [fix atc](#) instance
- mesh create_elementset = name of the AtC sub-command
- id = id to assign to the collection of FE nodes
- <xmin> <xmax> <ymin> <ymax> <zmin> <zmax> = coordinates of the bounding box that contains only the desired elements

Examples

```
fix_modify AtC mesh create_elementset middle -4.1 4.1 -100 100 -100 1100
```

Description

Command to assign an id to a set of FE elements to be used subsequently in defining material and mesh-based operations.

Restrictions

Only viable for rectangular grids.

Related AtC commands

- [fix_modify AtC command overview](#)
- [fix_modify AtC mesh delete_elements](#)
- [fix_modify AtC mesh nodeset_to_elementset](#)

Default

Coordinates are assumed to be in lattice units.

2.95.39 fix_modify AtC mesh create_faceset box command

Syntax

```
fix_modify <AtC fixID> mesh create_faceset <id> box <xmin> <xmax> <ymin> <ymax> <zmin>
           <zmax> <in|out> [units]
```

- AtC fixID = ID of [fix atc](#) instance
- mesh create_faceset = name of the AtC sub-command
- id = id to assign to the collection of FE faces
- box = use bounding box to define FE faces
- <xmin> <xmax> <ymin> <ymax> <zmin> <zmax> = coordinates of the bounding box that is coincident with the desired FE faces
- <in|out> = “in” gives inner faces to the box, “out” gives the outer faces to the box
- units = option to specify real as opposed to lattice units

Examples

```
fix_modify AtC mesh create_faceset obndy box -4.0 4.0 -12 12 -12 12 out
```

Description

Command to assign an id to a set of FE faces.

Restrictions

Only viable for rectangular grids.

Related AtC commands

- [fix_modify AtC command overview](#)
- [fix_modify AtC mesh create_faceset plane](#)

Default

The default options are units = lattice and the use of outer faces.

2.95.40 fix_modify AtC mesh create_faceset plane command

Syntax

```
fix_modify <AtC fixID> mesh create_faceset <id> plane <x|y|z> <val1> <x|y|z> <lval2> <uval2>
  ↳[units]
```

- AtC fixID = ID of [fix atc](#) instance
- mesh create_faceset = name of the AtC sub-command
- id = id to assign to the collection of FE faces
- plane = use plane to define faceset
- <val1>,<lval2>,<uval2> = plane is specified as the x|y|z=val1 plane bounded by the segments x|y|z = [lval2,uval2]
- units = option to specify real as opposed to lattice units

Examples

```
fix_modify AtC mesh create_faceset xyplane plane y 0 x -4 0
```

Description

Command to assign an id to a set of FE faces.

Restrictions

Only viable for rectangular grids.

Related AtC commands

- [fix_modify AtC command overview](#)
- [fix_modify AtC mesh create_faceset box](#)

Default

The default options are units = lattice.

2.95.41 fix_modify AtC mesh create_nodeset command

Syntax

```
fix_modify <AtC fixID> mesh create_nodeset <id> <xmin> <xmax> <ymin> <ymax> <zmin>  
      ↳<zmax>
```

- AtC fixID = ID of *fix atc* instance
- mesh create_nodeset = name of the AtC sub-command
- id = id to assign to the collection of FE nodes
- <xmin> <xmax> <ymin> <ymax> <zmin> <zmax> = coordinates of the bounding box that contains only the desired nodes

Examples

```
fix_modify AtC mesh create_nodeset lbc -12.1 -11.9 -12 12 -12 12
```

Description

Command to assign an id to a set of FE nodes to be used subsequently in defining boundary conditions.

Restrictions

None

Related AtC commands

- *fix_modify AtC command overview*
- *fix_modify AtC mesh add_to_nodeset*

Default

Coordinates are assumed to be in lattice units.

2.95.42 fix_modify AtC mesh delete_elements command

Syntax

```
fix_modify <AtC fixID> mesh delete_elements <id>
```

- AtC fixID = ID of *fix atc* instance
- mesh create_elementset = name of the AtC sub-command
- id = id of the element set

Examples

```
fix_modify AtC mesh delete_elements gap
```

Description

Deletes a group of elements from the mesh.

Restrictions

None.

Related AtC commands

- [fix_modify AtC command overview](#)
- [fix_modify AtC mesh create_elementset](#)
- [fix_modify AtC mesh nodeset_to_elementset](#)

Default

None.

2.95.43 fix_modify AtC mesh nodeset_to_elementset command

Syntax

```
fix_modify <AtC fixID> mesh nodeset_to_elementset <nodeset_id> <elementset_id> <max/min>
```

- AtC fixID = ID of [fix atc](#) instance
- mesh nodeset_to_elementset = name of the AtC sub-command
- nodeset_id = id of desired nodeset from which to create the elementset
- elementset_id = id to assign to the collection of FE elements
- <max/min> = flag to choose either the maximal or minimal elementset

Examples

```
fix_modify AtC mesh nodeset_to_elementset myNodeset myElementset min
```

Description

Command to create an elementset from an existing nodeset. Either the minimal element set of elements with all nodes in the set, or maximal element set with all elements with at least one node in the set, can be created.

Restrictions

None.

Related AtC commands

- [*fix_modify AtC command overview*](#)
- [*fix_modify AtC mesh create_elementset*](#)
- [*fix_modify AtC mesh delete_elements*](#)

Default

Unless specified, the maximal element set is created.

2.95.44 fix_modify AtC mesh output command

Syntax

```
fix_modify <AtC fixID> mesh output <file_prefix>
```

- AtC fixID = ID of [*fix atc*](#) instance
- mesh output = name of the AtC sub-command
- file_prefix = prefix of various generated output files

Examples

```
fix_modify AtC mesh output meshData
```

Description

Command to output mesh and associated data: nodesets, facesets, and elementssets. This data is only output once upon initialization since currently the mesh is static. Creates binary (EnSight, “gold” format) output of mesh data.

Restrictions

None.

Related AtC commands

- *fix_modify AtC command overview*

Default

None.

2.95.45 fix_modify AtC mesh quadrature command

Syntax

```
fix_modify <AtC fixID> mesh quadrature <quad>
```

- AtC fixID = ID of *fix atc* instance
- mesh quadrature = name of the AtC sub-command
- quad = *nodal* or *gauss1* or *gauss2* or *gauss3* or *face*

Examples

```
fix_modify AtC mesh quadrature face
```

Description

(Re-)assigns the quadrature style for an existing mesh. When a mesh is created its quadrature method defaults to gauss2. Use this call to change it after the fact.

Restrictions

None.

Related AtC commands

- *fix_modify AtC command overview*
- *fix_modify AtC mesh create*

Default

None.

2.95.46 fix_modify AtC mesh read command

Syntax

```
fix_modify <AtC fixID> mesh read <f|p> <f|p> <f|p>
```

- AtC fixID = ID of *fix atc* instance
- mesh read = name of the AtC sub-command
- filename = name of the file containing the mesh to be read
- f or p = periodicity flags for x-, y-, and z-direction (optional)

Examples

```
fix_modify AtC mesh read myComponent.mesh p p p  
fix_modify AtC mesh read myOtherComponent.exo
```

Description

Reads a mesh from a text or exodus file, and assigns periodic boundary conditions if needed.

Restrictions

None

Related AtC commands

- *fix_modify AtC command overview*
- *fix_modify AtC mesh create*
- *fix_modify AtC mesh write*

Default

Periodicity flags are set to false (f) by default.

2.95.47 fix_modify AtC mesh write command

Syntax

```
fix_modify <AtC fixID> mesh write <f|p> <f|p> <f|p>
```

- AtC fixID = ID of *fix atc* instance
- mesh write = name of the AtC sub-command
- filename = name of the file containing the mesh to be write

Examples

```
fix_modify AtC mesh write myMesh.mesh
```

Description

Writes a mesh to a text file.

Restrictions

None

Related AtC commands

- *fix_modify AtC command overview*
- *fix_modify AtC mesh create*
- *fix_modify AtC mesh read*

Default

None.

2.95.48 fix_modify AtC output command

Syntax

```
fix_modify <AtC fixID> output <filename_prefix> <frequency> [text|full_text|binary|vector_
-components|tensor_components]
fix_modify <AtC fixID> output index [step|time]
```

- AtC fixID = ID of *fix atc* instance
- *output* or *output index* = name of the AtC sub-command
- *filename_prefix* = prefix for data files (for *output*)
- *frequency* = frequency of output in timesteps (for *output*)

- optional keywords for *output*:
 - `text` = creates text output of index, step and nodal variable values for unique nodes
 - `full_text` = creates text output index, nodal id, step, nodal coordinates and nodal variable values for unique and image nodes
 - `binary` = creates binary EnSight output
 - `vector_components` = outputs vectors as scalar components
 - `tensor_components` = outputs tensor as scalar components (for use with ParaView)
- `step` or `time` = index output by step or by time (for *output index*)

Examples

```
fix_modify AtC output heatFE 100
fix_modify AtC output hardyFE 1 text tensor_components
fix_modify AtC output hardyFE 10 text binary tensor_components
fix_modify AtC output index step
```

Description

Creates text and/or binary (EnSight, “gold” format) output of nodal/mesh data which is transfer/physics specific. Output indexing by step or time is possible.

Restrictions

None.

Related AtC commands

- [fix_modify AtC command overview](#)
- [fix atc command](#)

Default

No default format. Output indexed by time.

2.95.49 fix_modify AtC output boundary_integral command

Syntax

```
fix_modify <AtC fixID> output boundary_integral <fieldname> faceset [name]
```

- AtC fixID = ID of [fix atc](#) instance
- output boundary_integral = name of the AtC sub-command
- fieldname = name of hardy field

- faceset = required keyword
- name= name of faceset

Examples

```
fix_modify AtC output boundary_integral stress faceset loop1
```

Description

Calculates a surface integral of the given field dotted with the outward normal of the faces and puts output in the “GLOBALS” file.

Restrictions

Must be used with the hardy/field type of [fix atc](#)

Related AtC commands

- [fix_modify AtC command overview](#)

Default

None.

2.95.50 fix_modify AtC output contour_integral command

Syntax

```
fix_modify <AtC fixID> output contour_integral <fieldname> faceset <name> [axis [x|y|z]]
```

- AtC fixID = ID of [fix atc](#) instance
- output contour_integral = name of the AtC sub-command
- fieldname = name of hardy field
- faceset = required keyword
- name = name of faceset
- axis *x* or axis *y* or axis *z* = (optional)

Examples

```
fix_modify AtC output contour_integral stress faceset loop1
```

Description

Calculates a surface integral of the given field dotted with the outward normal of the faces and puts output in the “GLOBALS” file.

Restrictions

Must be used with the hardy/field type of [fix atc](#)

Related AtC commands

- [fix_modify AtC command overview](#)

Default

None.

2.95.51 fix_modify AtC output nodeset command

Syntax

```
fix_modify <AtC fixID> output nodeset <nodeset_name> <operation>
```

- AtC fixID = ID of [fix atc](#) instance
- output nodeset = name of the AtC sub-command
- nodeset_name= name of nodeset to be operated on
- operation = *sum*
 - *sum* = creates nodal sum over nodes in specified nodeset

Examples

```
fix_modify AtC output nodeset nset1 sum
```

Description

Performs operation over the nodes belonging to specified nodeset and outputs resulting variable values to GLOBALS file.

Restrictions

None.

Related AtC commands

- [fix_modify AtC command overview](#)
- [fix atc command](#)

Default

None.

2.95.52 fix_modify AtC output volume_integral command

Syntax

```
fix_modify <AtC fixID> output volume_integral <elementset_name> <field>
```

- AtC fixID = ID of [fix atc](#) instance
- output volume_integral = name of the AtC sub-command
- elementset_name= name of elementset to be integrated over
- fieldname = name of field to integrate

Examples

```
fix_modify AtC output volume_integral eset1 mass_density
```

Description

Performs volume integration of specified field over elementset and outputs resulting variable values to GLOBALS file.

Restrictions

None.

Related AtC commands

- [fix_modify AtC command overview](#)
- [fix atc command](#)

Default

None.

2.95.53 fix_modify AtC pair_interactions command

2.95.54 fix_modify AtC bond_interactions command

Syntax

```
fix_modify <AtC fixID> pair_interactions <on|off>
fix_modify <AtC fixID> bond_interactions <on|off>
```

- AtC fixID = ID of [fix atc](#) instance
- *pair_interactions* or *bond_interactions* = name of the AtC sub-command
- *on* or *off* = activate or deactivate

Examples

```
fix_modify AtC pair_interactions off
fix_modify AtC bond_interactions on
```

Description

Include bonds and/or pairs in stress and heat flux computations.

Restrictions

None.

Related AtC commands

- [fix_modify AtC command overview](#)

Default

pair_interactions: on, *bond_interactions*: off

2.95.55 fix_modify AtC poisson_solver command

Syntax

```
fix_modify <AtC fixID> poisson_solver mesh create <nx> <ny> <nz> <region-ID> <f|p> <f|p>
  ↪<f|p>
```

- AtC fixID = ID of [fix atc](#) instance
- poisson_solver = name of the AtC sub-command
- *nx ny nz* = number of elements in x, y, and z
- region-id = id of region to be meshed
- *f* or *p* = periodicity flags for x, y, and z

Examples

```
fix_modify AtC poisson_solver mesh create 10 1 1 feRegion p p p
```

Description

Creates a uniform mesh in a rectangular region.

Restrictions

Creates only uniform rectangular grids in rectangular regions.

Related AtC commands

- [fix_modify AtC command overview](#)

Default

None.

2.95.56 fix_modify AtC read_restart command

Syntax

```
fix_modify <AtC fixID> read_restart <file_name>
```

- AtC fixID = ID of *fix atc* instance
- read_restart = name of the AtC sub-command
- file_name = name of AtC restart file

Examples

```
fix_modify AtC read_restart restart.mydata.AtC
```

Description

Reads the current state of the AtC fields from a named text-based restart file.

Restrictions

The restart file only contains fields and their time derivatives. The reference positions of the atoms and the commands that initialize the fix are not saved e.g. an identical mesh containing the same atoms will have to be recreated.

Related AtC commands

- *fix_modify AtC command overview*
- *fix_modify AtC write_restart*

Default

None.

2.95.57 fix_modify AtC remove_molecule command

Syntax

```
fix_modify <AtC fixID> remove_molecule <tag>
```

- AtC fixID = ID of *fix atc* instance
- remove_molecule = name of the AtC sub-command
- tag = tag for tracking a molecule

Examples

```
fix_modify AtC remove_molecule water
```

Description

Removes tag designated for tracking a specified set of molecules.

Restrictions

None.

Related AtC commands

- [fix_modify AtC command overview](#)
- [fix_modify AtC add_species](#)
- [fix_modify AtC add_molecule](#)
- [fix_modify AtC remove_species](#)

Default

None.

2.95.58 fix_modify AtC remove_source command

Syntax

```
fix_modify <AtC fixID> remove_source <field> <element_set>
```

- AtC fixID = ID of [fix atc](#) instance
- remove_source = name of the AtC sub-command
- field = field kind name valid for type of physics: *temperature* or *electron_temperature*
- element_set = name of set of elements

Examples

```
fix_modify AtC remove_source temperature groupNAME
```

Description

Remove a domain source.

Restrictions

The keyword *all* is reserved and thus not available as element_set name.

Related AtC commands

- [fix_modify AtC command overview](#)
- [fix_modify AtC source](#)

Default

None.

2.95.59 fix_modify AtC remove_species command

Syntax

```
fix_modify <AtC fixID> remove_species <tag>
```

- AtC fixID = ID of [fix atc](#) instance
- remove_species = name of the AtC sub-command
- tag = tag for tracking a species

Examples

```
fix_modify AtC remove_species gold
```

Description

Removes tag designated for tracking a specified species.

Restrictions

None.

Related AtC commands

- [*fix_modify AtC command overview*](#)
- [*fix_modify AtC add_species*](#)
- [*fix_modify AtC add_molecule*](#)
- [*fix_modify AtC remove_molecule*](#)

Default

None.

2.95.60 fix_modify AtC reset_atomic_reference_positions command

Syntax

```
fix_modify <AtC fixID> reset_atomic_reference_positions
```

- AtC fixID = ID of [*fix atc*](#) instance
- reset_atomic_reference_positions = name of the AtC sub-command

Examples

```
fix_modify AtC reset_atomic_reference_positions
```

Description

Resets the atomic positions ATC uses to perform point to field operations. It can be used to use perfect lattice sites in ATC but a thermalized or deformed lattice in LAMMPS.

Restrictions

None.

Related AtC commands

- [*fix_modify AtC command overview*](#)

Default

None

2.95.61 fix_modify AtC reset_time command**Syntax**

```
fix_modify <AtC fixID> reset_time <value>
```

- AtC fixID = ID of *fix atc* instance
- reset_time = name of the AtC sub-command
- value = new time value

Examples

```
fix_modify AtC reset_time 0.0
```

Description

Resets the simulation time counter.

Restrictions

None.

Related AtC commands

- *fix_modify AtC command overview*

Default

None

2.95.62 fix_modify AtC sample_frequency command**Syntax**

```
fix_modify <AtC fixID> sample_frequency <freq>
```

- AtC fixID = ID of *fix atc* instance
- sample_frequency = name of the AtC sub-command
- freq = frequency to sample fields in number of steps

Examples

```
fix_modify AtC sample_frequency 10
```

Description

Specifies a frequency at which fields are computed for the case where time filters are being applied.

Restrictions

Must be used with [fix atc hardy](#) and is only relevant when time filters are being used.

Related AtC commands

- [fix_modify AtC command overview](#)

Default

None.

2.95.63 fix_modify AtC set reference_potential_energy command

Syntax

```
fix_modify <AtC fixID> set reference_potential_energy [<value|filename>]
```

- AtC fixID = ID of [fix atc](#) instance
- set reference_potential_energy = name of the AtC sub-command
- value = optional user specified zero point for PE in native LAMMPS energy units
- filename = optional user specified string for file of nodal PE values to be read-in

Examples

```
fix_modify AtC set reference_potential_energy
fix_modify AtC set reference_potential_energy -0.05
fix_modify AtC set reference_potential_energy myPEvalues
```

Description

Used to set various quantities for the post-processing algorithms. It sets the zero point for the potential energy density using the value provided for all nodes, or from the current configuration of the lattice if no value is provided, or values provided within the specified filename.

Restrictions

Must be used with [fix atc hardy](#) or [fix atc field](#).

Related AtC commands

- [fix_modify AtC command overview](#)

Default

Defaults to the LAMMPS zero point i.e. isolated atoms.

2.95.64 fix_modify AtC source command

Syntax

```
fix_modify <AtC fixID> source <field> <element_set> <value|function>
```

- AtC fixID = ID of [fix atc](#) instance
- source = name of the AtC sub-command
- field = field kind name valid for type of physics: *temperature* or *electron_temperature*
- element_set = name of set of elements
- value or function = value or name of function followed by its parameters

Examples

```
fix_modify AtC source temperature middle temporal_ramp 10.0 0.0
```

Description

Add domain sources to the mesh. The units are consistent with LAMMPS's units for mass, length and time and are defined by the PDE being solved, e.g. for thermal transfer the balance equation is for energy and source is energy per time.

Restrictions

The keyword *all* is reserved and thus not available as element_set name.

Related AtC commands

- [fix_modify AtC command overview](#)
- [fix_modify AtC remove_source](#)

Default

None.

2.95.65 fix_modify AtC source_integration command

Syntax

```
fix_modify <AtC fixID> source_integration <fe|atom>
```

- AtC fixID = ID of [fix atc](#) instance
- source_integration = name of the AtC sub-command
- *fe* or *atom* = (undocumented)

Examples

```
fix_modify AtC source_integration atom
```

Description

(undocumented)

Restrictions

None.

Related AtC commands

- [fix_modify AtC command overview](#)

Default

Default is *fe*

2.95.66 fix_modify AtC temperature_definition command

Syntax

```
fix_modify <AtC fixID> temperature_definition <kinetic|total>
```

- AtC fixID = ID of *fix atc* instance
- temperature_definition = name of the AtC sub-command
- *kinetic* or *total* = (undocumented)

Examples

```
fix_modify AtC temperature_definition kinetic
```

Description

Change the definition for the atomic temperature used to create the finite element temperature. The kinetic option is based only on the kinetic energy of the atoms while the total option uses the total energy (kinetic + potential) of an atom.

Restrictions

This command is only valid when using thermal coupling. Also, while not a formal restriction, the user should ensure that associating a potential energy with each atom makes physical sense for the total option to be meaningful.

Related AtC commands

- *fix_modify AtC command overview*

Default

kinetic

2.95.67 fix_modify AtC filter command

Syntax

```
fix_modify <AtC fixID> filter <on|off|equilibrate>
```

- AtC fixID = ID of *fix atc* instance
- filter = name of the AtC sub-command

- *on* or *off* or *equilibrate* = Select state of filter

Examples

```
fix_modify AtC filter on
```

Description

Filters the MD dynamics to construct a more appropriate continuous field. Equilibrating first filters the time derivatives without changing the dynamics to provide a better initial condition to the filtered dynamics.

Restrictions

Only for use with these specific transfers: thermal, two_temperature

Related AtC commands

- [fix_modify AtC command overview](#)
- [fix_modify AtC filter scale](#)
- [fix_modify AtC equilibrium_start](#)

Default

off

2.95.68 fix_modify AtC time_integration command

Syntax

```
fix_modify <AtC fixID> time_integration <descriptor>
```

- AtC fixID = ID of [fix atc](#) instance
- time_integration = name of the AtC sub-command
- descriptor = *gear* or *fractional_step* or *verlet*

Examples

```
fix_modify AtC time_integration fractional_step
```

Description

Command to select the thermal or momentum time integration.

Options for thermal time integration:

gear

atomic velocity update with second order Verlet, nodal temperature update with third or fourth order Gear, thermostats based on controlling power

fractional_step

atomic velocity update with second order Verlet, mixed nodal temperature update, 3/4 Gear for continuum and 2 Verlet for atomic contributions, thermostats based on controlling discrete energy changes

Options for momentum time integration:

verlet

atomic velocity update with second order Verlet, nodal temperature update with second order Verlet, kinetostats based on controlling force

fractional_step

atomic velocity update with second order Verlet, mixed nodal momentum update, second order Verlet for continuum and exact second order Verlet for atomic contributions, kinetostats based on controlling discrete momentum changes

gear

atomic velocity update with second order Verlet, nodal temperature update with third or fourth order Gear, kinetostats based on controlling power.

Restrictions

None.

Related AtC commands

- *fix_modify AtC command overview*

Default

None.

2.95.69 fix_modify AtC track_displacement command

Syntax

```
fix_modify <AtC fixID> track_displacement <on|off>
```

- AtC fixID = ID of *fix atc* instance
- track_displacement = name of the AtC sub-command
- *on* or *off* = (undocumented)

Examples

```
fix_modify AtC track_displacement on
```

Description

Determines whether displacement is tracked or not. For solids problems this is a useful quantity, but for fluids it is not relevant.

Restrictions

Some constitutive models require the displacement field.

Related AtC commands

- *fix_modify AtC command overview*

Default

on

2.95.70 fix_modify AtC unfix command

Syntax

```
fix_modify <AtC fixID> unfix <field> <nodeset>
```

- AtC fixID = ID of *fix atc* instance
- unfix = name of the AtC sub-command
- field = field kind name valid for type of physics: *temperature* or *electron_temperature*
- nodeset = name of set of nodes to apply boundary condition

Examples

```
fix_modify AtC unfix temperature groupNAME
```

Description

Removes constraint on field values for specified nodes.

Restrictions

The keyword *all* is reserved and thus not available as nodeset name.

Related AtC commands

- [fix_modify AtC command overview](#)
- [fix_modify AtC fix](#)

Default

None.

2.95.71 fix_modify AtC unfix_flux command

Syntax

```
fix_modify <AtC fixID> unfix_flux <field> <face_set> <value|function>
```

- AtC fixID = ID of [fix atc](#) instance
- unfix_flux = name of the AtC sub-command
- field = field kind name valid for type of physics: *temperature* or *electron_temperature*
- face_set = name of set of element faces

Examples

```
fix_modify AtC unfix_flux temperature faceSet
```

Description

Command for removing prescribed normal fluxes e.g. heat_flux, stress.

Restrictions

None.

Related AtC commands

- [fix_modify AtC command overview](#)
- [fix_modify AtC fix_flux](#)

Default

None.

2.95.72 fix_modify AtC write_atom_weights command**Syntax**

```
fix_modify <AtC fixID> write_atom_weights <filename> <frequency>
```

- AtC fixID = ID of *fix atc* instance
- write_atom_weights = name of the AtC sub-command
- filename = name of file that atomic weights are written to
- frequency = how often writes will occur

Examples

```
fix_modify AtC write_atom_weights atm_wt_file.txt 10
```

Description

Command for writing the values of atomic weights to a specified file.

Restrictions

None.

Related AtC commands

- *fix_modify AtC command overview*

Default

None

2.95.73 fix_modify AtC write_restart command**Syntax**

```
fix_modify <AtC fixID> write_restart <file_name>
```

- AtC fixID = ID of *fix atc* instance
- write_restart = name of the AtC sub-command
- file_name = name of AtC restart file

Examples

```
fix_modify AtC write_restart restart.mydata.AtC
```

Description

Dumps the current state of the fields to a named text-based restart file. This done when the command is invoked and not repeated, unlike the otherwise similar LAMMPS command.

Restrictions

None.

Related AtC commands

- [fix_modify AtC command overview](#)
- [fix_modify AtC read_restart](#)

Default

None.

2.96 fix mol/swap command

2.96.1 Syntax

```
fix ID group-ID mol/swap N X itype jtype seed T keyword value ...
```

- ID, group-ID are documented in [fix](#) command
- atom/swap = style name of this fix command
- N = invoke this fix every N steps
- X = number of swaps to attempt every N steps
- itype,jtype = two atom types (1-Ntypes or type label) to swap with each other
- seed = random # seed (positive integer)
- T = scaling temperature of the MC swaps (temperature units)
- zero or more keyword/value pairs may be appended to args
- keyword = *ke*

ke value = no or yes

no = no conservation of kinetic energy after atom swaps

yes = kinetic energy is conserved after atom swaps

2.96.2 Examples

```
fix 2 all mol/swap 100 1 2 3 29494 300.0 ke no
fix mySwap fluid mol/swap 500 10 1 2 482798 1.0

labelmap atom 1 A 2 B
fix mySwap fluid mol/swap 500 10 A B 482798 1.0
```

2.96.3 Description

This fix performs Monte Carlo swaps of two specified atom types within a randomly selected molecule. Two possible use cases are as follows.

First, consider a mixture of some molecules with atoms of *itype* and other molecules with atoms of *jtype*. The fix will select a random molecule and attempt to swap all the *itype* atoms to *jtype* for the first kind of molecule, or all the *jtype* atoms to *itype* for the second kind. Because the swap will only take place if it is energetically favorable, the fix can be used to determine the miscibility of 2 different kinds of molecules much more quickly than just dynamics would do it.

Second, consider diblock co-polymers with two types of monomers *itype* and *jtype*. The fix will select a random molecule and attempt to do a *itype* \leftrightarrow *jtype* swap of all those monomers within the molecule. Thus the fix can be used to find the energetically favorable fractions of two flavors of diblock co-polymers.

Intra-molecular swaps of atom types are attempted every *N* timesteps. On that timestep, *X* swaps are attempted. For each attempt a single molecule ID is randomly selected. The range of possible molecule IDs from *loID* to *hiID* is pre-computed before each run begins. The *loID/hiID* is set for the molecule with the smallest/largest ID which has any *itype* or *jtype* atoms in it. Note that if you define a system with many molecule IDs between *loID* and *hiID* which have no *itype* or *jtype* atoms, then the fix will be inefficient at performing swaps. Also note that if atoms with molecule ID = 0 exist, they are not considered molecules by this fix; they are assumed to be solvent atoms or molecules.

Candidate atoms for swapping must also be in the fix group. Atoms within the selected molecule which are not *itype* or *jtype* are ignored.

When an atom is swapped from *itype* to *jtype* (or vice versa), if charges are defined, the charge values for *itype* versus *jtype* atoms are also swapped. This requires that all *itype* atoms in the system have the same charge value. Likewise all *jtype* atoms in the system must have the same charge value. If this is not the case, LAMMPS issues a warning that it cannot swap charge values.

If the *ke* keyword is set to yes, which is the default, and the masses of *itype* and *jtype* atoms are different, then when a swap occurs, the velocity of the swapped atom is rescaled by the sqrt of the mass ratio, so as to conserve the kinetic energy of the atom.

The potential energy of the entire system is computed before and after each swap is performed within a single molecule. The specified temperature *T* is used in the Metropolis criterion to accept or reject the attempted swap. If the swap is rejected all swapped values are reversed.

The potential energy calculations can include systems and models with the following features:

- manybody pair styles, including EAM
- hybrid pair styles
- long-range electrostatics (kspace)
- triclinic systems
- potential energy contributions from other fixes

For the last bullet point, fixes can have an associated potential energy. Examples of such fixes include: [efield](#), [gravity](#), [addforce](#), [langevin](#), [restrain](#), [temp/berendsen](#), [temp/rescale](#), and [wall fixes](#). For that energy to be included in the total potential energy of the system (the quantity used for the swap accept/reject decision), you MUST enable the [fix_modify energy](#) option for that fix. The doc pages for individual [fix](#) commands specify if this should be done.

Note

One comment on computational efficiency. If the cutoff lengths defined for the pair style are different for itype versus jtype atoms (for any of their interactions with any other atom type), then a new neighbor list needs to be generated for every attempted swap. This is potentially expensive if N is small or X is large.

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

This fix writes the state of the fix to [binary restart files](#). This includes information about the random number generator seed, the next timestep for MC exchanges, the number of exchange attempts and successes etc. See the [read_restart](#) command for info on how to re-specify a fix in an input script that reads a restart file, so that the operation of the fix continues in an uninterrupted fashion.

Note

For this to work correctly, the timestep must **not** be changed after reading the restart with [reset_timestep](#). The fix will try to detect it and stop with an error.

None of the [fix_modify](#) options are relevant to this fix.

This fix computes a global vector of length 2, which can be accessed by various [output commands](#). The vector values are the following global cumulative quantities:

1. swap attempts
2. swap accepts

The vector values calculated by this fix are “intensive”.

No parameter of this fix can be used with the *start/stop* keywords of the [run](#) command. This fix is not invoked during [energy minimization](#).

2.96.5 Restrictions

This fix is part of the MC package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) doc page for more info.

2.96.6 Related commands

[fix atom/swap](#), [fix gcmc](#)

2.96.7 Default

The option default is ke = yes.

2.97 fix momentum command

Accelerator Variants: *momentum/kk*

2.98 fix momentum/chunk command

2.98.1 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

```
fix ID group-ID momentum/chunk N chunkID keyword values ...
```

- ID, group-ID are documented in [fix](#) command
- momentum/chunk = style name of this fix command
- N = adjust the momentum per chunk every this many timesteps
- chunkID = ID of [compute chunk/atom](#) command
one or more keyword/value settings may be appended to each of the fix commands:
- keyword = *linear* or *angular* or *rescale*
linear values = xflag yflag zflag
xflag,yflag,zflag = 0/1 to exclude/include each dimension
angular values = none
rescale values = none

2.98.2 Examples

```
fix 1 all momentum 1 linear 1 1 0
fix 1 all momentum 1 linear 1 1 1 rescale
fix 1 all momentum 100 linear 1 1 1 angular
fix 1 all momentum/chunk 100 molchunk linear 1 1 1 angular
```

2.98.3 Description

Fix momentum zeroes the linear and/or angular momentum of the group of atoms every N timesteps by adjusting the velocities of the atoms. Fix momentum/chunk works equivalently, but operates on a per-chunk basis.

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 or chunk 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 ensure 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).

The *rescale* keyword enables conserving the kinetic energy of the group or chunk of atoms by rescaling the velocities after the momentum was removed.

Note that the [velocity](#) command can be used to create initial velocities with zero aggregate linear and/or angular momentum.

Styles with a *gpu*, *intel*, *kk*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed on the [Accelerator packages](#) page. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the GPU, INTEL, KOKKOS, OPENMP, and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the [Build package](#) page for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the [-suffix command-line switch](#) when you invoke LAMMPS, or you can use the [suffix](#) command in your input script.

See the [Accelerator packages](#) page for more instructions on how to use the accelerated styles effectively.

2.98.4 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](#).

2.98.5 Restrictions

Fix momentum/chunk is part of the EXTRA-FIX package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

2.98.6 Related commands

fix recenter, velocity

2.98.7 Default

none

2.99 fix move command

2.99.1 Syntax

```
fix ID group-ID move style args keyword values ...
```

- ID, group-ID are documented in [fix](#) command
- move = style name of this fix command
- style = *linear* or *wiggle* or *rotate* or *transrot* or *variable*

linear args = Vx Vy Vz

Vx,Vy,Vz = components of velocity vector (velocity units), any component can be specified as [NULL](#)

wiggle args = Ax Ay Az period

Ax,Ay,Az = components of amplitude vector (distance units), any component can be specified as [NULL](#)

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)

transrot args = Vx Vy Vz Px Py Pz Rx Ry Rz period

Vx,Vy,Vz = components of velocity vector (velocity units)

Px,Py,Pz = origin point of axis of rotation (distance units)

Rx,Ry,Rz = axis of rotation vector

period = period of rotation (time units)

variable args = v_dx v_dy v_dz v_vx v_vy v_vz

v_dx,v_dy,v_dz = 3 variable names that calculate x,y,z displacement as function of time, any [component](#) can be specified as [NULL](#)

v_vx,v_vy,v_vz = 3 variable names that calculate x,y,z velocity as function of time, any [component](#) can be specified as [NULL](#)

- zero or more keyword/value pairs may be appended

- keyword = *units*

units value = box or lattice

2.99.2 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
fix 3 boundary move transrot 0.1 0.1 0.0 0.0 0.0 0.0 0.0 1.0 5.0 units box
```

2.99.3 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.

Note

The atoms affected by this fix should not normally be time integrated by other fixes (e.g. [fix nve](#), [fix nvt](#)), since that will change their positions and velocities twice.

Note

As atoms move due to this fix, they will pass through periodic boundaries and be remapped to the other side of the simulation box, just as they would during normal time integration (e.g. via the [fix nve](#) command). It is up to you to decide whether periodic boundaries are appropriate with the kind of atom motion you are prescribing with this fix.

Note

As discussed 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

$$X(t) = X_0 + V * \text{delta}$$

where $X_0 = (x_0, y_0, z_0)$ is their position at the time the fix is specified, V is the specified velocity vector with components (V_x, V_y, V_z) , and delta is the time elapsed since the fix was specified. This style also sets the velocity of each atom to $V = (V_x, V_y, V_z)$. 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

$$X(t) = X_0 + A \sin(\omega * \text{delta})$$

where $X_0 = (x_0, y_0, z_0)$ is their position at the time the fix is specified, A is the specified amplitude vector with components (A_x, A_y, A_z), ω is $2\pi / \text{period}$, 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 $R = (R_x, R_y, R_z)$ that goes through a point $P = (P_x, P_y, P_z)$. The *period* of the rotation is also specified. The direction of rotation for the atoms 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.

This style also sets the velocity of each atom to $(\omega \times R_{\perp})$ where ω is its angular velocity around the rotation axis and R_{\perp} is a perpendicular vector from the rotation axis to the atom. If the defined *atom_style* assigns an angular velocity or angular momentum or orientation to each atom (*atom styles* sphere, ellipsoid, line, tri, body), then those properties are also updated appropriately to correspond to the atom's motion and rotation over time.

The *transrot* style combines the effects of *rotate* and *linear* so that it is possible to prescribe a rotating group of atoms that also moves at a constant velocity. The arguments are for the translation first and then for the rotation. Since the rotation affects all coordinate components, it is not possible to set any of the translation vector components to NULL.

The *variable* style allows the position and velocity components of each atom to be set by formulas specified via the [variable](#) command. Each of the 6 variables is specified as an argument to the fix as `v_name`, where `name` is the variable name that is defined elsewhere in the input script.

Each variable must be of either the *equal* or *atom* style. *Equal*-style variables compute a single numeric quantity, that can be a function of the timestep as well as of other simulation values. *Atom*-style variables compute a numeric quantity for each atom, that can be a function per-atom quantities, such as the atom's position, as well as of the timestep and other simulation values. Note that this fix stores the original coordinates of each atom (see note below) so that per-atom quantity can be used in an atom-style variable formula. See the [variable](#) command for details.

The first 3 variables (`v_dx, v_dy, v_dz`) specified for the *variable* style are used to calculate a displacement from the atom's original position at the time the fix was specified. The second 3 variables (`v_vx, v_vy, v_vz`) specified are used to compute a velocity for each atom.

Any of the 6 variables can be specified as NULL. If both the displacement and velocity variables for a particular x,y,z component are specified as NULL, then the position and velocity of that component is time integrated the same as the [fix nve](#) command would perform, using the corresponding force component on the atom. If only the velocity variable for a component is specified as NULL, then the displacement variable will be used to set the position of the atom, and its velocity component will not be changed. If only the displacement variable for a component is specified as NULL, then the velocity variable will be used to set the velocity of the atom, and the position of the atom will be time integrated using that velocity.

The *units* keyword determines the meaning of the distance units used to define the *linear* velocity and *wiggle* amplitude and *rotate* origin. This setting is ignored for the *variable* style. A *box* value selects standard units as defined by the

units command, e.g. velocity in Angstroms/fs 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.

2.99.4 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.

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*.

For *rRESPA time integration*, this fix adjusts the position and velocity of atoms on the outermost rRESPA level.

2.99.5 Restrictions

none

2.99.6 Related commands

fix nve, *displace_atoms*

2.99.7 Default

none

The option default is units = lattice.

2.100 fix msst command

2.100.1 Syntax

```
fix ID group-ID msst dir shockvel keyword value ...
```

- ID, group-ID are documented in [fix](#) command
- msst = style name of this fix
- dir = *x* or *y* or *z*
- shockvel = shock velocity (strictly positive, distance/time units)
- zero or more keyword value pairs may be appended
- keyword = *q* or *mu* or *p0* or *v0* or *e0* or *tscale* or *beta* or *dftb*

q value = cell mass-like parameter (mass²/distance⁴ units)

mu value = artificial viscosity (mass/length/time units)

p0 value = initial pressure in the shock equations (pressure units)

v0 value = initial simulation cell volume in the shock equations (distance³ units)

e0 value = initial total energy (energy units)

tscale value = reduction in initial temperature (unitless fraction between 0.0 and 1.0)

dftb value = yes or no for whether using MSST in conjunction with DFTB+

beta value = scale factor for improved energy conservation

2.100.2 Examples

```
fix 1 all msst y 100.0 q 1.0e5 mu 1.0e5
fix 2 all msst z 50.0 q 1.0e4 mu 1.0e4 v0 4.3419e+03 p0 3.7797e+03 e0 -9.72360e+02 tscale 0.01
fix 1 all msst y 100.0 q 1.0e5 mu 1.0e5 dftb yes beta 0.5
```

2.100.3 Description

This command performs the Multi-Scale Shock Technique (MSST) integration to update positions and velocities each timestep to mimic a compressive shock wave passing over the system. See ([Reed](#)) for a detailed description of this method. The MSST varies the cell volume and temperature in such a way as to restrain the system to the shock Hugoniot and the Rayleigh line. These restraints correspond to the macroscopic conservation laws dictated by a shock front. *shockvel* determines the steady shock velocity that will be simulated.

To perform a simulation, choose a value of *q* that provides volume compression on the timescale of 100 fs to 1 ps. If the volume is not compressing, either the shock speed is chosen to be below the material sound speed or *p0* has been chosen inaccurately. Volume compression at the start can be sped up by using a non-zero value of *tscale*. Use the smallest value of *tscale* that results in compression.

Under some special high-symmetry conditions, the pressure (volume) and/or temperature of the system may oscillate for many cycles even with an appropriate choice of mass-like parameter *q*. Such oscillations have physical significance in some cases. The optional *mu* keyword adds an artificial viscosity that helps break the system symmetry to equilibrate to the shock Hugoniot and Rayleigh line more rapidly in such cases.

The keyword *tscale* is a factor between 0 and 1 that determines what fraction of thermal kinetic energy is converted to compressive strain kinetic energy at the start of the simulation. Setting this parameter to a non-zero value may assist in compression at the start of simulations where it is slow to occur.

If keywords $e0$, $p0$, or $v0$ are not supplied, these quantities will be calculated on the first step, after the energy specified by $tscal$ is removed. The value of $e0$ is not used in the dynamical equations, but is used in calculating the deviation from the Hugoniot.

The keyword β is a scaling term that can be added to the MSST ionic equations of motion to account for drift in the conserved quantity during long timescale simulations, similar to a Berendsen thermostat. See ([Reed](#)) and ([Goldman](#)) for more details. The value of β must be between 0.0 and 1.0 inclusive. A value of 0.0 means no contribution, a value of 1.0 means a full contribution.

Values of shockvel less than a critical value determined by the material response will not have compressive solutions. This will be reflected in lack of significant change of the volume in the MSST.

For all pressure styles, the simulation box stays orthogonal in shape. Parrinello-Rahman boundary conditions (tilted box) are supported by LAMMPS, but are not implemented for MSST.

This fix computes a temperature and pressure and potential energy each timestep. To do this, the fix creates its own computes of style “temp” “pressure”, and “pe”, as if these commands had been issued:

```
compute fix-ID_MSST_temp all temp
compute fix-ID_MSST_press all pressure fix-ID_MSST_temp

compute fix-ID_MSST_pe all pe
```

See the [compute temp](#) and [compute pressure](#) commands for details. Note that the IDs of the new computes are the fix-ID + “_MSST_temp” or “_MSST_press” or “_MSST_pe”. The group for the new computes is “all”.

The $dftb$ keyword is to allow this fix to be used when LAMMPS is being driven by DFTB+, a density-functional tight-binding code. If the keyword $dftb$ is used with a value of yes, then the MSST equations are altered to account for the electron entropy contribution to the Hugonio relations and total energy. See ([Reed2](#)) and ([Goldman](#)) for details on this contribution. In this case, you must define a [fix external](#) command in your input script, which is used to callback to DFTB+ during the LAMMPS timestepping. DFTB+ will communicate its info to LAMMPS via that fix.

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

This fix writes the state of all internal variables to [binary restart files](#). See the [read_restart](#) command for info on how to re-specify a fix in an input script that reads a restart file, so that the operation of the fix continues in an uninterrupted fashion.

The cumulative energy change in the system imposed by this fix is included in the [thermodynamic output](#) keywords $ecouple$ and $econserve$. See the [thermo_style](#) doc page for details.

This fix computes a global scalar which can be accessed by various [output commands](#). The scalar is the same cumulative energy change due to this fix described in the previous paragraph. The scalar value calculated by this fix is “extensive”.

The progress of the MSST can be monitored by printing the global scalar and global vector quantities computed by the fix.

As mentioned above, the scalar is the cumulative energy change due to the fix. By monitoring the thermodynamic $econserve$ output, this can be used to test if the MD timestep is sufficiently small for accurate integration of the dynamic equations.

The global vector contains four values in the following order. The vector values output by this fix are “intensive”.

[$dhugoniot$, $drayleigh$, $lagrangian_speed$, $lagrangian_position$]

1. $dhugoniot$ is the departure from the Hugoniot (temperature units).

2. *drayleight* is the departure from the Rayleigh line (pressure units).
3. *lagrangian_speed* is the laboratory-frame Lagrangian speed (particle velocity) of the computational cell (velocity units).
4. *lagrangian_position* is the computational cell position in the reference frame moving at the shock speed. This is usually a good estimate of distance of the computational cell behind the shock front.

To print these quantities to the log file with descriptive column headers, the following LAMMPS commands are suggested:

```
fix      msst all msst z
variable dhug equal f_msst[1]
variable dray equal f_msst[2]
variable lgr_vel equal f_msst[3]
variable lgr_pos equal f_msst[4]
thermo_style custom step temp ke pe lz pzz econserve v_dhug v_dray v_lgr_vel v_lgr_pos f_msst
```

2.100.5 Restrictions

This fix style is part of the SHOCK package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

All cell dimensions must be periodic. This fix can not be used with a triclinic cell. The MSST fix has been tested only for the group-ID all.

2.100.6 Related commands

fix nphug, fix deform

2.100.7 Default

The keyword defaults are q = 10, mu = 0, tscale = 0.01, dftb = no, beta = 0.0. Note that p0, v0, and e0 are calculated on the first timestep.

(Reed) Reed, Fried, and Joannopoulos, Phys. Rev. Lett., 90, 235503 (2003).

(Reed2) Reed, J. Phys. Chem. C, 116, 2205 (2012).

(Goldman) Goldman, Srinivasan, Hamel, Fried, Gaus, and Elstner, J. Phys. Chem. C, 117, 7885 (2013).

2.101 fix mvv/dpd command

2.102 fix mvv/edpd command

2.103 fix mvv/tdpd command

2.103.1 Syntax

```
fix ID group-ID mvv/dpd lambda
fix ID group-ID mvv/edpd lambda
fix ID group-ID mvv/tdpd lambda
```

- ID, group-ID are documented in [fix](#) command
- mvv/dpd, mvv/edpd, mvv/tdpd = style name of this fix command
- lambda = (optional) relaxation parameter (unitless)

2.103.2 Examples

```
fix 1 all mvv/dpd
fix 1 all mvv/dpd 0.5
fix 1 all mvv/edpd
fix 1 all mvv/edpd 0.5
fix 1 all mvv/tdpd
fix 1 all mvv/tdpd 0.5
```

2.103.3 Description

Perform time integration using the modified velocity-Verlet (MVV) algorithm to update position and velocity (fix mvv/dpd), or position, velocity and temperature (fix mvv/edpd), or position, velocity and concentration (fix mvv/tdpd) for particles in the group each timestep.

The modified velocity-Verlet (MVV) algorithm aims to improve the stability of the time integrator by using an extrapolated version of the velocity for the force evaluation:

$$\begin{aligned} v(t + \frac{\Delta t}{2}) &= v(t) + \frac{\Delta t}{2} \cdot a(t) \\ r(t + \Delta t) &= r(t) + \Delta t \cdot v(t + \frac{\Delta t}{2}) \\ a(t + \Delta t) &= \frac{1}{m} \cdot F [r(t + \Delta t), v(t) + \lambda \cdot \Delta t \cdot a(t)] \\ v(t + \Delta t) &= v(t + \frac{\Delta t}{2}) + \frac{\Delta t}{2} \cdot a(t + \Delta t) \end{aligned}$$

where the parameter λ depends on the specific choice of DPD parameters, and needs to be tuned on a case-by-case basis. Specification of a *lambda* value is optional. If specified, the setting must be from 0.0 to 1.0. If not specified, a default value of 0.5 is used, which effectively reproduces the standard velocity-Verlet (VV) scheme. For more details, see [Groot](#).

Fix *mvv/dpd* updates the position and velocity of each atom. It can be used with the [pair_style mdpd](#) command or other pair styles such as [pair dpd](#).

Fix *mvv/edpd* updates the per-atom temperature, in addition to position and velocity, and must be used with the [pair_style edpd](#) command.

Fix *mvv/tdpd* updates the per-atom chemical concentration, in addition to position and velocity, and must be used with the [pair_style tdpd](#) command.

2.103.4 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](#).

2.103.5 Restrictions

These fixes are part of the DPD-MESO package. They are only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

Changed in version 29Aug2024.

This fix is incompatible with deformation controls that remap velocity, for instance the *remap v* option of [fix deform](#).

2.103.6 Related commands

[pair_style mdpd](#), [pair_style edpd](#), [pair_style tdpd](#)

2.103.7 Default

The default value for the optional *lambda* parameter is 0.5.

(Groot) Groot and Warren, J Chem Phys, 107: 4423-4435 (1997). DOI: 10.1063/1.474784

2.104 fix neb command

2.104.1 Syntax

`fix ID group-ID neb Kspring keyword value`

- ID, group-ID are documented in [fix](#) command
- neb = style name of this fix command
- Kspring = spring constant for parallel nudging force (force/distance units or force units, see parallel keyword)
- zero or more keyword/value pairs may be appended
- keyword = *parallel* or *perp* or *end*

parallel value = neigh or ideal or equal
 neigh = parallel nudging force based on distance to neighbor replicas (Kspring = force/distance units)
 ideal = parallel nudging force based on interpolated ideal position (Kspring = force units)
 equal = parallel nudging force based on interpolated ideal position before climbing, then interpolated
 ↴ideal energy whilst climbing (Kspring = force units)
 perp value = Kspring2
 Kspring2 = spring constant for perpendicular nudging force (force/distance units)
 end values = estyle Kspring3
 estyle = first or last or last/efirst or last/efirst/middle
 first = apply force to first replica
 last = apply force to last replica
 last/efirst = apply force to last replica and set its target energy to that of first replica
 last/efirst/middle = same as last/efirst plus prevent middle replicas having lower energy than first
 ↴replica
 Kspring3 = spring constant for target energy term (1/distance units)

2.104.2 Examples

```
fix 1 active neb 10.0
fix 2 all neb 1.0 perp 1.0 end last
fix 2 all neb 1.0 perp 1.0 end first 1.0 end last 1.0
fix 1 all neb 1.0 parallel ideal end last/efirst 1
```

2.104.3 Description

Add nudging forces to atoms in the group for a multi-replica simulation run via the [neb](#) command to perform a nudged elastic band (NEB) calculation for finding the transition state. Hi-level explanations of NEB are given with the [neb](#) command and on the [Howto replica](#) doc page. The fix neb command must be used with the “neb” command and defines how inter-replica nudging forces are computed. A NEB calculation is divided in two stages. In the first stage n replicas are relaxed toward a MEP until convergence. In the second stage, the climbing image scheme (see ([Henkelman2](#))) is enabled, so that the replica having the highest energy relaxes toward the saddle point (i.e. the point of highest energy along the MEP), and a second relaxation is performed.

A key purpose of the nudging forces is to keep the replicas equally spaced. During the NEB calculation, the $3N$ -length vector of interatomic force $F_i = -\nabla V$ for each replica i is altered. For all intermediate replicas (i.e. for $1 < i < N$, except the climbing replica) the force vector becomes:

$$F_i = -\nabla V + (\nabla V \cdot T')T' + F_{\parallel} + F_{\perp}$$

T' is the unit “tangent” vector for replica i and is a function of R_i, R_{i-1}, R_{i+1} , and the potential energy of the 3 replicas; it points roughly in the direction of $R_{i+1} - R_{i-1}$; see the ([Henkelman1](#)) paper for details. R_i are the atomic coordinates of replica i ; R_{i-1} and R_{i+1} are the coordinates of its neighbor replicas. The term $\nabla V \cdot T'$ is used to remove the component of the gradient parallel to the path which would tend to distribute the replica unevenly along the path. F_{\parallel} is an artificial nudging force which is applied only in the tangent direction and which maintains the equal spacing between replicas (see below for more information). F_{\perp} is an optional artificial spring which is applied in a direction perpendicular to the tangent direction and which prevent the paths from forming acute kinks (see below for more information).

In the second stage of the NEB calculation, the interatomic force F_i for the climbing replica (the replica of highest energy after the first stage) is changed to:

$$F_i = -\nabla V + 2(\nabla V \cdot T')T' + F_{\perp}$$

and the relaxation procedure is continued to a new converged MEP.

The keyword *parallel* specifies how the parallel nudging force is computed. With a value of *neigh*, the parallel nudging force is computed as in ([Henkelman1](#)) by connecting each intermediate replica with the previous and the next image:

$$F_{\parallel} = K_{spring} \cdot (|R_{i+1} - R_i| - |R_i - R_{i-1}|)$$

Note that in this case the specified *Kspring* is in force/distance units.

With a value of *ideal*, the spring force is computed as suggested in ([WeinanE](#))

$$F_{\parallel} = -K_{spring} \cdot (RD - RD_{ideal}) / (2 \cdot meanDist)$$

where *RD* is the “reaction coordinate” see *neb* section, and *RD_{ideal}* is the ideal *RD* for which all the images are equally spaced. I.e. $RD_{ideal} = (i - 1) \cdot meanDist$ when the climbing replica is off, where *i* is the replica number). The *meanDist* is the average distance between replicas. Note that in this case the specified *Kspring* is in force units. When the climbing replica is on, *RD_{ideal}* and *meanDist* are calculated separately each side of the climbing image. Note that the *ideal* form of nudging can often be more effective at keeping the replicas equally spaced before climbing, then equally spaced either side of the climbing image whilst climbing.

With a value of *equal* the spring force is computed as for *ideal* when the climbing replica is off, promoting equidistance. When the climbing replica is on, the spring force is computed to promote equidistant absolute differences in energy, rather than distance, each side of the climbing image:

$$F_{\parallel} = -K_{spring} \cdot (ED - ED_{ideal}) / (2 \cdot meanEDist)$$

where *ED* is the cumulative sum of absolute energy differences:

$$ED = \sum_{i < N} |E(R_{i+1}) - E(R_i)|,$$

meanEDist is the average absolute energy difference between replicas up to the climbing image or from the climbing image to the final image, for images before or after the climbing image respectively. *ED_{ideal}* is the corresponding cumulative sum of average absolute energy differences in each case, in close analogy to *ideal*. This form of nudging is to aid schemes which integrate forces along, or near to, NEB pathways such as [fix_pafi](#).

The keyword *perp* specifies if and how a perpendicular nudging force is computed. It adds a spring force perpendicular to the path in order to prevent the path from becoming too strongly kinked. It can significantly improve the convergence of the NEB calculation when the resolution is poor. I.e. when few replicas are used; see ([Maras](#)) for details.

The perpendicular spring force is given by

$$F_{\perp} = K_{spring2} \cdot F(R_{i-1}, R_i, R_{i+1})(R_{i+1} + R_{i-1} - 2R_i)$$

where *Kspring2* is the specified value. $F(R_{i-1}, R_i, R_{i+1})$ is a smooth scalar function of the angle $R_{i-1}R_iR_{i+1}$. It is equal to 0.0 when the path is straight and is equal to 1 when the angle $R_{i-1}R_iR_{i+1}$ is acute. $F(R_{i-1}, R_i, R_{i+1})$ is defined in ([Jonsson](#)).

If *Kspring2* is set to 0.0 (the default) then no perpendicular spring force is added.

By default, no additional forces act on the first and last replicas during the NEB relaxation, so these replicas simply relax toward their respective local minima. By using the key word *end*, additional forces can be applied to the first and/or last replicas, to enable them to relax toward a MEP while constraining their energy *E* to the target energy *ETarget*.

If $E_{Target} > E$, the interatomic force F_i for the specified replica becomes:

$$\begin{aligned} F_i &= -\nabla V + (\nabla V \cdot T' + (E - E_{Target}) \cdot K_{spring3})T', && \text{when } \nabla V \cdot T' < 0 \\ F_i &= -\nabla V + (\nabla V \cdot T' + (E_{Target} - E) \cdot K_{spring3})T', && \text{when } \nabla V \cdot T' > 0 \end{aligned}$$

The “spring” constant on the difference in energies is the specified *Kspring3* value.

When *estyle* is specified as *first*, the force is applied to the first replica. When *estyle* is specified as *last*, the force is applied to the last replica. Note that the *end* keyword can be used twice to add forces to both the first and last replicas.

For both these *estyle* settings, the target energy *ETarget* is set to the initial energy of the replica (at the start of the NEB calculation).

If the *estyle* is specified as *last/efirst* or *last/efirst/middle*, force is applied to the last replica, but the target energy *ETarget* is continuously set to the energy of the first replica, as it evolves during the NEB relaxation.

The difference between these two *estyle* options is as follows. When *estyle* is specified as *last/efirst*, no change is made to the inter-replica force applied to the intermediate replicas (neither first or last). If the initial path is too far from the MEP, an intermediate replica may relax “faster” and reach a lower energy than the last replica. In this case the intermediate replica will be relaxing toward its own local minima. This behavior can be prevented by specifying *estyle* as *last/efirst/middle* which will alter the inter-replica force applied to intermediate replicas by removing the contribution of the gradient to the inter-replica force. This will only be done if a particular intermediate replica has a lower energy than the first replica. This should effectively prevent the intermediate replicas from over-relaxing.

After converging a NEB calculation using an *estyle* of *last/efirst/middle*, you should check that all intermediate replicas have a larger energy than the first replica. If this is not the case, the path is probably not a MEP.

Finally, note that the last replica may never reach the target energy if it is stuck in a local minima which has a larger energy than the target energy.

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

No information about this fix is written to *binary restart files*. None of the *fix_modify* options are relevant to this fix. No global or per-atom quantities are stored by this fix for access by various *output commands*. No parameter of this fix can be used with the *start/stop* keywords of the *run* command.

The forces due to this fix are imposed during an energy minimization, as invoked by the *minimize* command via the *neb* command.

2.104.5 Restrictions

This command can only be used if LAMMPS was built with the REPLICA package. See the *Build package* doc page for more info.

2.104.6 Related commands

neb

2.104.7 Default

The option defaults are parallel = neigh, perp = 0.0, ends is not specified (no inter-replica force on the end replicas).

(Henkelman1) Henkelman and Jonsson, J Chem Phys, 113, 9978-9985 (2000).

(Henkelman2) Henkelman, Uberuaga, Jonsson, J Chem Phys, 113, 9901-9904 (2000).

(WeinanE) E, Ren, Vanden-Eijnden, Phys Rev B, 66, 052301 (2002).

(Jonsson) Jonsson, Mills and Jacobsen, in Classical and Quantum Dynamics in Condensed Phase Simulations, edited by Berne, Ciccotti, and Coker World Scientific, Singapore, 1998, p 385.

(Maras) Maras, Trushin, Stukowski, Ala-Nissila, Jonsson, Comp Phys Comm, 205, 13-21 (2016).

2.105 fix neb/spin command

2.105.1 Syntax

```
fix ID group-ID neb/spin Kspring
```

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

Kspring = spring constant for parallel nudging force
(force/distance units or force units, see parallel keyword)

2.105.2 Examples

```
fix 1 active neb/spin 1.0
```

2.105.3 Description

Add nudging forces to spins in the group for a multi-replica simulation run via the [neb/spin](#) command to perform a geodesic nudged elastic band (GNEB) calculation for finding the transition state. Hi-level explanations of GNEB are given with the [neb/spin](#) command and on the [Howto replica](#) doc page. The fix neb/spin command must be used with the “neb/spin” command and defines how inter-replica nudging forces are computed. A GNEB calculation is divided in two stages. In the first stage n replicas are relaxed toward a MEP until convergence. In the second stage, the climbing image scheme is enabled, so that the replica having the highest energy relaxes toward the saddle point (i.e. the point of highest energy along the MEP), and a second relaxation is performed.

The nudging forces are calculated as explained in ([Bessarab](#)). See this reference for more explanation about their expression.

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

No information about this fix is written to [binary restart files](#). None of the [fix_modify](#) options are relevant to this fix. No global or per-atom quantities are stored by this fix for access by various [output commands](#). No parameter of this fix can be used with the [start/stop](#) keywords of the [run](#) command.

The forces due to this fix are imposed during an energy minimization, as invoked by the [minimize](#) command via the [neb/spin](#) command.

2.105.5 Restrictions

This command can only be used if LAMMPS was built with the SPIN package. See the *Build package* doc page for more info.

2.105.6 Related commands

neb_spin

2.105.7 Default

none

(Bessarab) Bessarab, Uzdin, Jonsson, Comp Phys Comm, 196, 335-347 (2015).

2.106 fix nvt command

Accelerator Variants: *nvt/gpu*, *nvt/intel*, *nvt/kk*, *nvt/omp*

2.107 fix npt command

Accelerator Variants: *npt/gpu*, *npt/intel*, *npt/kk*, *npt/omp*

2.108 fix nph command

Accelerator Variants: *nph/kk*, *nph/omp*

2.108.1 Syntax

```
fix ID group-ID style_name keyword value ...
```

- ID, group-ID are documented in *fix* command
- style_name = *nvt* or *npt* or *nph*
- one or more keyword/value pairs may be appended

keyword = temp or iso or aniso or tri or x or y or z or xy or yz or xz or couple or tchain or pchain

→ or mtk or tloop or ploop or nreset or drag or ptemp or dilate or scalexy or scaleyz or scalexz or
→ flip or fixedpoint or update

temp values = Tstart Tstop Tdamp

Tstart,Tstop = external temperature at start/end of run

Tdamp = temperature damping parameter (time units)

iso or aniso or tri values = Pstart Pstop Pdamp

Pstart,Pstop = scalar external pressure at start/end of run (pressure units)

Pdamp = pressure damping parameter (time units)

x or y or z or xy or yz or xz values = Pstart Pstop Pdamp
 Pstart,Pstop = external stress tensor component at start/end of run (pressure units)
 Pdamp = stress damping parameter (time units)
 couple = none or xyz or xy or yz or xz
 tchain value = N
 N = length of thermostat chain (1 = single thermostat)
 pchain value = N
 N length of thermostat chain on barostat (0 = no thermostat)
 mtk value = yes or no = add in MTK adjustment term or not
 tloop value = M
 M = number of sub-cycles to perform on thermostat
 ploop value = M
 M = number of sub-cycles to perform on barostat thermostat
 nreset value = reset reference cell every this many timesteps
 drag value = Df
 Df = drag factor added to barostat/thermostat (0.0 = no drag)
 ptemp value = Ttarget
 Ttarget = target temperature for barostat
 dilate value = dilate-group-ID
 dilate-group-ID = only dilate atoms in this group due to barostat volume changes
 scalexy value = yes or no = scale xy with ly
 scaleyz value = yes or no = scale yz with lz
 scalexz value = yes or no = scale xz with lz
 flip value = yes or no = allow or disallow box flips when it becomes highly skewed
 fixedpoint values = x y z
 x,y,z = perform barostat dilation/contraction around this point (distance units)
 update value = dipole or dipole/dlm
 dipole = update dipole orientation (only for sphere variants)
 dipole/dlm = use DLM integrator to update dipole orientation (only for sphere variants)

2.108.2 Examples

```
fix 1 all nvt temp 300.0 300.0 100.0
fix 1 water npt temp 300.0 300.0 100.0 iso 0.0 0.0 1000.0
fix 2 jello npt temp 300.0 300.0 100.0 tri 5.0 5.0 1000.0
fix 2 ice nph x 1.0 1.0 0.5 y 2.0 2.0 0.5 z 3.0 3.0 0.5 yz 0.1 0.1 0.5 xz 0.2 0.2 0.5 xy 0.3 0.3 0.5 nreset 1000
```

2.108.3 Description

These commands perform time integration on Nose-Hoover style non-Hamiltonian equations of motion which are designed to generate positions and velocities sampled from the canonical (nvt), isothermal-isobaric (npt), and isenthalpic (nph) ensembles. This updates the position and velocity for atoms in the group each timestep.

The thermostating and barostatting is achieved by adding some dynamic variables which are coupled to the particle velocities (thermostating) and simulation domain dimensions (barostatting). In addition to basic thermostating and barostatting, these fixes can also create a chain of thermostats coupled to the particle thermostat, and another chain of thermostats coupled to the barostat variables. The barostat can be coupled to the overall box volume, or to individual dimensions, including the xy, xz and yz tilt dimensions. The external pressure of the barostat can be specified as either a scalar pressure (isobaric ensemble) or as components of a symmetric stress tensor (constant stress ensemble). When used correctly, the time-averaged temperature and stress tensor of the particles will match the target values specified by Tstart/Tstop and Pstart/Pstop.

The equations of motion used are those of Shinoda et al in ([Shinoda](#)), which combine the hydrostatic equations of Martyna, Tobias and Klein in ([Martyna](#)) with the strain energy proposed by Parrinello and Rahman in ([Parrinello](#)). The time integration schemes closely follow the time-reversible measure-preserving Verlet and rRESPA integrators derived by Tuckerman et al in ([Tuckerman](#)).

The thermostat parameters for fix styles *nvt* and *npt* are specified using the *temp* keyword. Other thermostat-related keywords are *tchain*, *tloop* and *drag*, which are discussed below.

The thermostat is applied to only the translational degrees of freedom for the particles. The translational degrees of freedom can also have a bias velocity removed before thermostating takes place; see the description below. The desired temperature at each timestep is a ramped value during the run from *Tstart* to *Tstop*. The *Tdamp* parameter is specified in time units and determines how rapidly the temperature is relaxed. For example, a value of 10.0 means to relax the temperature in a timespan of (roughly) 10 time units (e.g. τ or fs or ps - see the [units](#) command). The atoms in the fix group are the only ones whose velocities and positions are updated by the velocity/position update portion of the integration.

Note

A Nose-Hoover thermostat will not work well for arbitrary values of *Tdamp*. If *Tdamp* is too small, the temperature can fluctuate wildly; if it is too large, the temperature will take a very long time to equilibrate. A good choice for many models is a *Tdamp* of around 100 timesteps. Note that this is NOT the same as 100 time units for most [units](#) settings. A simple way to ensure this, is via using an *immediate variable* expression accessing the thermo property ‘dt’, which is the length of the time step. Example:

```
fix 1 all nvt temp 300.0 300.0 $(100.0*dt)
```

The barostat parameters for fix styles *npt* and *nph* is specified using one or more of the *iso*, *aniso*, *tri*, *x*, *y*, *z*, *xy*, *xz*, *yz*, and *couple* keywords. These keywords give you the ability to specify all 6 components of an external stress tensor, and to couple various of these components together so that the dimensions they represent are varied together during a constant-pressure simulation.

Other barostat-related keywords are *pchain*, *mtk*, *ploop*, *nreset*, *drag*, and *dilate*, which are discussed below.

Orthogonal simulation boxes have 3 adjustable dimensions (x,y,z). Triclinic (non-orthogonal) simulation boxes have 6 adjustable dimensions (x,y,z,xy,xz,yz). The *create_box*, *read data*, and *read_restart* commands specify whether the simulation box is orthogonal or non-orthogonal (triclinic) and explain the meaning of the xy,xz,yz tilt factors.

The target pressures for each of the 6 components of the stress tensor can be specified independently via the *x*, *y*, *z*, *xy*, *xz*, *yz* keywords, which correspond to the 6 simulation box dimensions. For each component, the external pressure or tensor component at each timestep is a ramped value during the run from *Pstart* to *Pstop*. If a target pressure is specified for a component, then the corresponding box dimension will change during a simulation. For example, if the *y* keyword is used, the *y*-box length will change. If the *xy* keyword is used, the *xy* tilt factor will change. A box dimension will not change if that component is not specified, although you have the option to change that dimension via the [fix deform](#) command.

Note that in order to use the *xy*, *xz*, or *yz* keywords, the simulation box must be triclinic, even if its initial tilt factors are 0.0.

For all barostat keywords, the *Pdamp* parameter operates like the *Tdamp* parameter, determining the time scale on which pressure is relaxed. For example, a value of 10.0 means to relax the pressure in a timespan of (roughly) 10 time units (e.g. τ or fs or ps - see the [units](#) command).

Note

A Nose-Hoover barostat will not work well for arbitrary values of *Pdamp*. If *Pdamp* is too small, the pressure and volume can fluctuate wildly; if it is too large, the pressure will take a very long time to equilibrate. A good choice for many models is a *Pdamp* of around 1000 timesteps. However, note that *Pdamp* is specified in time units, and that timesteps are NOT the same as time units for most *units* settings.

The relaxation rate of the barostat is set by its inertia *W*:

$$W = (N + 1)k_B T_{\text{target}} P_{\text{damp}}^2$$

where *N* is the number of atoms, k_B is the Boltzmann constant, and T_{target} is the target temperature of the barostat (*Martyna*). If a thermostat is defined, T_{target} is the target temperature of the thermostat. If a thermostat is not defined, T_{target} is set to the current temperature of the system when the barostat is initialized. If this temperature is too low the simulation will quit with an error. Note: in previous versions of LAMMPS, T_{target} would default to a value of 1.0 for *lj* units and 300.0 otherwise if the system had a temperature of exactly zero.

If a thermostat is not specified by this fix, T_{target} can be manually specified using the *Ptemp* parameter. This may be useful if the barostat is initialized when the current temperature does not reflect the steady state temperature of the system. This keyword may also be useful in athermal simulations where the temperature is not well defined.

Regardless of what atoms are in the fix group (the only atoms which are time integrated), a global pressure or stress tensor is computed for all atoms. Similarly, when the size of the simulation box is changed, all atoms are re-scaled to new positions, unless the keyword *dilate* is specified with a *dilate-group-ID* for a group that represents a subset of the atoms. This can be useful, for example, to leave the coordinates of atoms in a solid substrate unchanged and controlling the pressure of a surrounding fluid. This option should be used with care, since it can be unphysical to dilate some atoms and not others, because it can introduce large, instantaneous displacements between a pair of atoms (one dilated, one not) that are far from the dilation origin. Also note that for atoms not in the fix group, a separate time integration fix like *fix nve* or *fix nvt* can be used on them, independent of whether they are dilated or not.

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 P_{xx} and P_{zz} components of the stress tensor are coupled. *Xyz* means all 3 diagonal components are coupled. Coupling means two things: the instantaneous stress will be computed as an average of the corresponding diagonal components, and the coupled box dimensions will be changed together in lockstep, meaning coupled dimensions will be dilated or contracted by the same percentage every timestep. The *Pstart*, *Pstop*, *Pdamp* parameters for any coupled dimensions must be identical. *Couple xyz* can be used for a 2d simulation; the *z* dimension is simply ignored.

The *iso*, *aniso*, and *tri* keywords are simply shortcuts that are equivalent to specifying several other keywords together.

The keyword *iso* means couple all 3 diagonal components together when pressure is computed (hydrostatic pressure), and dilate/contract the dimensions together. Using “*iso Pstart Pstop Pdamp*” is the same as specifying these 4 keywords:

```
x Pstart Pstop Pdamp
y Pstart Pstop Pdamp
z Pstart Pstop Pdamp
couple xyz
```

The keyword *aniso* means *x*, *y*, and *z* dimensions are controlled independently using the P_{xx} , P_{yy} , and P_{zz} components of the stress tensor as the driving forces, and the specified scalar external pressure. Using “*aniso Pstart Pstop Pdamp*” is the same as specifying these 4 keywords:

```
x Pstart Pstop Pdamp
y Pstart Pstop Pdamp
z Pstart Pstop Pdamp
couple none
```

The keyword *tri* means *x*, *y*, *z*, *xy*, *xz*, and *yz* dimensions are controlled independently using their individual stress components as the driving forces, and the specified scalar pressure as the external normal stress. Using “*tri Pstart Pstop Pdamp*” is the same as specifying these 7 keywords:

```
x Pstart Pstop Pdamp
y Pstart Pstop Pdamp
z Pstart Pstop Pdamp
xy 0.0 0.0 Pdamp
yz 0.0 0.0 Pdamp
xz 0.0 0.0 Pdamp
couple none
```

In some cases (e.g. for solids) the pressure (volume) and/or temperature of the system can oscillate undesirably when a Nose/Hoover barostat and thermostat is applied. The optional *drag* keyword will damp these oscillations, although it alters the Nose/Hoover equations. A value of 0.0 (no drag) leaves the Nose/Hoover formalism unchanged. A non-zero value adds a drag term; the larger the value specified, the greater the damping effect. Performing a short run and monitoring the pressure and temperature is the best way to determine if the drag term is working. Typically a value between 0.2 to 2.0 is sufficient to damp oscillations after a few periods. Note that use of the drag keyword will interfere with energy conservation and will also change the distribution of positions and velocities so that they do not correspond to the nominal NVT, NPT, or NPH ensembles.

An alternative way to control initial oscillations is to use chain thermostats. The keyword *tchain* determines the number of thermostats in the particle thermostat. A value of 1 corresponds to the original Nose-Hoover thermostat. The keyword *pchain* specifies the number of thermostats in the chain thermostating the barostat degrees of freedom. A value of 0 corresponds to no thermostating of the barostat variables.

The *mtk* keyword controls whether or not the correction terms due to Martyna, Tuckerman, and Klein are included in the equations of motion ([Martyna](#)). Specifying *no* reproduces the original Hoover barostat, whose volume probability distribution function differs from the true NPT and NPH ensembles by a factor of $1/V$. Hence using *yes* is more correct, but in many cases the difference is negligible.

The keyword *tloop* can be used to improve the accuracy of integration scheme at little extra cost. The initial and final updates of the thermostat variables are broken up into *tloop* sub-steps, each of length $dt/tloop$. This corresponds to using a first-order Suzuki-Yoshida scheme ([Tuckerman](#)). The keyword *ploop* does the same thing for the barostat thermostat.

The keyword *nreset* controls how often the reference dimensions used to define the strain energy are reset. If this keyword is not used, or is given a value of zero, then the reference dimensions are set to those of the initial simulation domain and are never changed. If the simulation domain changes significantly during the simulation, then the final average pressure tensor will differ significantly from the specified values of the external stress tensor. A value of *nstep* means that every *nstep* timesteps, the reference dimensions are set to those of the current simulation domain.

The *scaleyz*, *scalexz*, and *scalexy* keywords control whether or not the corresponding tilt factors are scaled with the associated box dimensions when barostatting triclinic periodic cells. The default values *yes* will turn on scaling, which corresponds to adjusting the linear dimensions of the cell while preserving its shape. Choosing *no* ensures that the tilt factors are not scaled with the box dimensions. See below for restrictions and default values in different situations. In older versions of LAMMPS, scaling of tilt factors was not performed. The old behavior can be recovered by setting all three scale keywords to *no*.

The *flip* keyword allows the tilt factors for a triclinic box to exceed half the distance of the parallel box length, as discussed below. 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 applied stress induces large deformations (e.g. in a liquid), this means the box shape can tilt dramatically and LAMMPS will run less efficiently, due to the large volume of communication needed to acquire ghost atoms around a processor's irregular-shaped subdomain. For extreme values of tilt, LAMMPS may also lose atoms and generate an error.

The *fixedpoint* keyword specifies the fixed point for barostat volume changes. 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 trajectories are unaffected by the chosen value, except for a time-dependent constant translation of positions.

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 finite-size particles, e.g. spheroids via use of the *atom_style hybrid sphere dipole* command.

The default dipole orientation integrator can be changed to the Dullweber-Leimkuhler-McLachlan integration scheme (*Dullweber*) when using *update* with the value *dipole/dlm*. This integrator is symplectic and time-reversible, giving better energy conservation and allows slightly longer timesteps at only a small additional computational cost.

Note

Using a barostat coupled to tilt dimensions *xy*, *xz*, *yz* can sometimes result in arbitrarily large values of the tilt dimensions, i.e. a dramatically deformed simulation box. LAMMPS allows the tilt factors to grow a small amount beyond the normal limit of half the box length (0.6 times the box length), and then performs a box “flip” to an equivalent periodic cell. See the discussion of the *flip* keyword above, to allow this bound to be exceeded, if desired.

The flip operation is described in more detail in the page for *fix deform*. Both the barostat dynamics and the atom trajectories are unaffected by this operation. However, if a tilt factor is incremented by a large amount (1.5 times the box length) on a single timestep, LAMMPS can not accommodate this event and will terminate the simulation with an error. This error typically indicates that there is something badly wrong with how the simulation was constructed, such as specifying values of *Pstart* that are too far from the current stress value, or specifying a timestep that is too large. Triclinic barostatting should be used with care. This also is true for other barostat styles, although they tend to be more forgiving of insults. In particular, it is important to recognize that equilibrium liquids can not support a shear stress and that equilibrium solids can not support shear stresses that exceed the yield stress.

One exception to this rule is if the first 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.

Note

Unlike the *fix temp/berendsen* command which performs thermostatting but NO time integration, these fixes perform thermostatting/barostatting AND time integration. Thus you should not use any other time integration fix, such as *fix nve* on atoms to which this fix is applied. Likewise, *fix nvt* and *fix npt* should not normally be used on atoms that also have their temperature controlled by another fix - e.g. by *fix langevin* or *fix temp/rescale* commands.

See the *Howto thermostat* and *Howto barostat* doc pages for a discussion of different ways to compute temperature and perform thermostatting and barostatting.

These fixes compute a temperature and pressure each timestep. To do this, the thermostat and barostat fixes create their own computes of style “temp” and “pressure”, as if one of these sets of commands had been issued:

For fix nvt:

```
compute fix-ID_temp group-ID temp
```

For fix npt and fix nph:

```
compute fix-ID_temp all temp
compute fix-ID_press all pressure fix-ID_temp
```

For fix nvt, the group for the new temperature compute is the same as the fix group. For fix npt and fix nph, the group for both the new temperature and pressure compute is “all” since pressure is computed for the entire system. In the case of fix nph, the temperature compute is not used for thermostating, but just for a kinetic-energy contribution to the pressure. 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”.

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 these fix’s temperature or pressure via the [compute_modify](#) command. Or you can print this temperature or pressure during thermodynamic output via the [thermo_style custom](#) command using the appropriate compute-ID. It also means that changing attributes of *thermo_temp* or *thermo_press* will have no effect on this fix.

Like other fixes that perform thermostating, this fix can be used with [compute commands](#) that remove a “bias” from the atom velocities. E.g. to apply the thermostat only to atoms within a spatial *region*, or to remove the center-of-mass velocity from a group of atoms, or to remove the x-component of velocity from the calculation.

This is not done by default, but only if the [fix_modify](#) command is used to assign a temperature compute to this fix that includes such a bias term. See the doc pages for individual [compute temp commands](#) to determine which ones include a bias. In this case, the thermostat works in the following manner: bias is removed from each atom, thermostating is performed on the remaining thermal degrees of freedom, and the bias is added back in.

These fixes can be used with either the *verlet* or *respa integrators*. When using one of the barostat fixes with *respa*, LAMMPS uses an integrator constructed according to the following factorization of the Liouville propagator (for two rRESPA levels):

$$\begin{aligned} \exp(iL\Delta t) = & \hat{E} \exp\left(iL_{T,\text{baro}} \frac{\Delta t}{2}\right) \exp\left(iL_{T,\text{part}} \frac{\Delta t}{2}\right) \exp\left(iL_{\epsilon,2} \frac{\Delta t}{2}\right) \exp\left(iL_2^{(2)} \frac{\Delta t}{2}\right) \\ & \times \left[\exp\left(iL_2^{(1)} \frac{\Delta t}{2n}\right) \exp\left(iL_{\epsilon,1} \frac{\Delta t}{2n}\right) \exp\left(iL_1 \frac{\Delta t}{n}\right) \exp\left(iL_{\epsilon,1} \frac{\Delta t}{2n}\right) \exp\left(iL_2^{(1)} \frac{\Delta t}{2n}\right) \right]^n \\ & \times \exp\left(iL_2^{(2)} \frac{\Delta t}{2}\right) \exp\left(iL_{\epsilon,2} \frac{\Delta t}{2}\right) \exp\left(iL_{T,\text{part}} \frac{\Delta t}{2}\right) \exp\left(iL_{T,\text{baro}} \frac{\Delta t}{2}\right) \\ & + \mathcal{O}(\Delta t^3) \end{aligned}$$

This factorization differs somewhat from that of Tuckerman et al, in that the barostat is only updated at the outermost rRESPA level, whereas Tuckerman’s factorization requires splitting the pressure into pieces corresponding to the forces computed at each rRESPA level. In theory, the latter method will exhibit better numerical stability. In practice, because Pdamp is normally chosen to be a large multiple of the outermost rRESPA timestep, the barostat dynamics are not the limiting factor for numerical stability. Both factorizations are time-reversible and can be shown to preserve the phase space measure of the underlying non-Hamiltonian equations of motion.

Note

This implementation has been shown to conserve linear momentum up to machine precision under NVT dynamics. Under NPT dynamics, for a system with zero initial total linear momentum, the total momentum fluctuates close to zero. It may occasionally undergo brief excursions to non-negligible values, before returning close to zero. Over long simulations, this has the effect of causing the center-of-mass to undergo a slow random walk. This can be mitigated by resetting the momentum at infrequent intervals using the [fix momentum](#) command.

The fix npt and fix nph commands can be used with rigid bodies or mixtures of rigid bodies and non-rigid particles (e.g. solvent). But there are also [fix rigid/npt](#) and [fix rigid/nph](#) commands, which are typically a more natural choice. See the page for those commands for more discussion of the various ways to do this.

Styles with a *gpu*, *intel*, *kk*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed on the [Accelerator packages](#) page. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the GPU, INTEL, KOKKOS, OPENMP, and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the [Build package](#) page for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the [-suffix command-line switch](#) when you invoke LAMMPS, or you can use the [suffix](#) command in your input script.

See the [Accelerator packages](#) page for more instructions on how to use the accelerated styles effectively.

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

These fixes writes the state of all the thermostat and barostat variables to [binary restart files](#). See the [read_restart](#) command for info on how to re-specify a fix in an input script that reads a restart file, so that the operation of the fix continues in an uninterrupted fashion.

The [fix_modify temp](#) and [press](#) options are supported by these fixes. You can use them to assign a [compute](#) you have defined to this fix which will be used in its thermostating or barostating procedure, as described above. If you do this, note that the kinetic energy derived from the compute temperature should be consistent with the virial term computed using all atoms for the pressure. LAMMPS will warn you if you choose to compute temperature on a subset of atoms.

Note

If both the *temp* and *press* keywords are used in a single thermo_modify command (or in two separate commands), then the order in which the keywords are specified is important. Note that a [pressure compute](#) defines its own temperature compute as an argument when it is specified. The *temp* keyword will override this (for the pressure compute being used by fix npt), but only if the *temp* keyword comes after the *press* keyword. If the *temp* keyword comes before the *press* keyword, then the new pressure compute specified by the *press* keyword will be unaffected by the *temp* setting.

The cumulative energy change in the system imposed by these fixes, via either thermostating and/or barostating, is included in the [thermodynamic output](#) keywords *ecouple* and *econserve*. See the [thermo_style](#) page for details.

These fixes compute a global scalar which can be accessed by various *output commands*. The scalar is the same cumulative energy change due to this fix described in the previous paragraph. The scalar value calculated by this fix is “extensive”.

These fixes compute also compute a global vector of quantities, which can be accessed by various *output commands*. The vector values are “intensive”.

The vector stores internal Nose/Hoover thermostat and barostat variables. The number and meaning of the vector values depends on which fix is used and the settings for keywords *tchain* and *pchain*, which specify the number of Nose/Hoover chains for the thermostat and barostat. If no thermostating is done, then *tchain* is 0. If no barostating is done, then *pchain* is 0. In the following list, “ndof” is 0, 1, 3, or 6, and is the number of degrees of freedom in the barostat. Its value is 0 if no barostat is used, else its value is 6 if any off-diagonal stress tensor component is barostatted, else its value is 1 if *couple xyz* is used or *couple xy* for a 2d simulation, otherwise its value is 3.

The order of values in the global vector and their meaning is as follows. The notation means there are *tchain* values for eta, followed by *tchain* for eta_dot, followed by ndof for omega, etc:

- eta[tchain] = particle thermostat displacements (unitless)
- eta_dot[tchain] = particle thermostat velocities (1/time units)
- omega[ndof] = barostat displacements (unitless)
- omega_dot[ndof] = barostat velocities (1/time units)
- etap[pchain] = barostat thermostat displacements (unitless)
- etap_dot[pchain] = barostat thermostat velocities (1/time units)
- PE_eta[tchain] = potential energy of each particle thermostat displacement (energy units)
- KE_eta_dot[tchain] = kinetic energy of each particle thermostat velocity (energy units)
- PE_omega[ndof] = potential energy of each barostat displacement (energy units)
- KE_omega_dot[ndof] = kinetic energy of each barostat velocity (energy units)
- PE_etap[pchain] = potential energy of each barostat thermostat displacement (energy units)
- KE_etap_dot[pchain] = kinetic energy of each barostat thermostat velocity (energy units)
- PE_strain[1] = scalar strain energy (energy units)

These fixes can ramp their external temperature and pressure over multiple runs, using the *start* and *stop* keywords of the *run* command. See the *run* command for details of how to do this.

These fixes are not invoked during *energy minimization*.

2.108.5 Restrictions

X, *y*, *z* cannot be barostatted if the associated dimension is not periodic. *Xy*, *xz*, and *yz* can only be barostatted if the simulation domain is triclinic and the second dimension in the keyword (*y* dimension in *xy*) is periodic. *Z*, *xz*, and *yz*, cannot be barostatted for 2D simulations. The *create_box*, *read data*, and *read_restart* commands specify whether the simulation box is orthogonal or non-orthogonal (triclinic) and explain the meaning of the *xy,xz,yz* tilt factors.

For the *temp* keyword, the final Tstop cannot be 0.0 since it would make the external T = 0.0 at some timestep during the simulation which is not allowed in the Nose/Hoover formulation.

The *scaleyz yes* and *scalexz yes* keyword/value pairs can not be used for 2D simulations. *scaleyz yes*, *scalexz yes*, and *scalexy yes* options can only be used if the second dimension in the keyword is periodic, and if the tilt factor is not coupled to the barostat via keywords *tri*, *yz*, *xz*, and *xy*.

These fixes can be used with dynamic groups as defined by the [group](#) command. Likewise they can be used with groups to which atoms are added or deleted over time, e.g. a deposition simulation. However, the conservation properties of the thermostat and barostat are defined for systems with a static set of atoms. You may observe odd behavior if the atoms in a group vary dramatically over time or the atom count becomes very small.

2.108.6 Related commands

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

2.108.7 Default

The keyword defaults are tchain = 3, pchain = 3, mtk = yes, tloop = 1, ploop = 1, nreset = 0, drag = 0.0, dilate = all, couple = none, flip = yes, scaleyz = scalexz = scalexy = yes if periodic in second dimension and not coupled to barostat, otherwise no.

(Martyna) Martyna, Tobias and Klein, J Chem Phys, 101, 4177 (1994).

(Parrinello) Parrinello and Rahman, J Appl Phys, 52, 7182 (1981).

(Tuckerman) Tuckerman, Alejandre, Lopez-Rendon, Jochim, and Martyna, J Phys A: Math Gen, 39, 5629 (2006).

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

(Dullweber) Dullweber, Leimkuhler and McLachlan, J Chem Phys, 107, 5840 (1997).

2.109 fix nvt/eff command

2.110 fix npt/eff command

2.111 fix nph/eff command

2.111.1 Syntax

```
fix ID group-ID style_name keyword value ...
```

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

- style_name = *nvt/eff* or *npt/eff* or *nph/eff*

one or more keyword value pairs may be appended

keyword = temp or iso or aniso or tri or x or y or z or xy or yz or xz or couple or tchain or pchain

or mtk or tloop or ploop or nreset or drag or dilate

temp values = Tstart Tstop Tdamp

Tstart,Tstop = external temperature at start/end of run

Tdamp = temperature damping parameter (time units)

iso or aniso or tri values = Pstart Pstop Pdamp

Pstart,Pstop = scalar external pressure at start/end of run (pressure units)

Pdamp = pressure damping parameter (time units)

x or y or z or xy or yz or xz values = Pstart Pstop Pdamp

Pstart,Pstop = external stress tensor component at start/end of run (pressure units)

Pdamp = stress damping parameter (time units)
 couple = none or xyz or xy or yz or xz
 tchain value = length of thermostat chain (1 = single thermostat)
 pchain values = length of thermostat chain on barostat (0 = no thermostat)
 mtk value = yes or no = add in MTK adjustment term or not
 tloop value = number of sub-cycles to perform on thermostat
 ploop value = number of sub-cycles to perform on barostat thermostat
 nreset value = reset reference cell every this many timesteps
 drag value = drag factor added to barostat/thermostat (0.0 = no drag)
 dilate value = all or partial

2.111.2 Examples

```

fix 1 all nvt/eff temp 300.0 300.0 0.1
fix 1 part npt/eff temp 300.0 300.0 0.1 iso 0.0 0.0 1.0
fix 2 part npt/eff temp 300.0 300.0 0.1 tri 5.0 5.0 1.0
fix 2 ice nph/eff x 1.0 1.0 0.5 y 2.0 2.0 0.5 z 3.0 3.0 0.5 yz 0.1 0.1 0.5 xz 0.2 0.2 0.5 xy 0.3 0.3 0.5 nreset
  ↵1000
  
```

2.111.3 Description

These commands perform time integration on Nose-Hoover style non-Hamiltonian equations of motion for nuclei and electrons in the group for the [electron force field](#) model. The fixes are designed to generate positions and velocities sampled from the canonical (nvt), isothermal-isobaric (npt), and isenthalpic (nph) ensembles. This is achieved by adding some dynamic variables which are coupled to the particle velocities (thermostating) and simulation domain dimensions (barostatting). In addition to basic thermostating and barostatting, these fixes can also create a chain of thermostats coupled to the particle thermostat, and another chain of thermostats coupled to the barostat variables. The barostat can be coupled to the overall box volume, or to individual dimensions, including the *xy*, *xz* and *yz* tilt dimensions. The external pressure of the barostat can be specified as either a scalar pressure (isobaric ensemble) or as components of a symmetric stress tensor (constant stress ensemble). When used correctly, the time-averaged temperature and stress tensor of the particles will match the target values specified by Tstart/Tstop and Pstart/Pstop.

The operation of these fixes is exactly like that described by the [fix nvt, npt, and nph](#) commands, except that the radius and radial velocity of electrons are also updated. Likewise the temperature and pressure calculated by the fix, using the computes it creates (as discussed in the [fix nvt, npt, and nph](#) doc page), are performed with computes that include the eFF contribution to the temperature or kinetic energy from the electron radial velocity.

Note

there are two different pressures that can be reported for eFF when defining the pair_style (see [pair eff/cut](#) to understand these settings), one (default) that considers electrons do not contribute radial virial components (i.e. electrons treated as incompressible ‘rigid’ spheres) and one that does. The radial electronic contributions to the virials are only tallied if the flexible pressure option is set, and this will affect both global and per-atom quantities. In principle, the true pressure of a system is somewhere in between the rigid and the flexible eFF pressures, but, for most cases, the difference between these two pressures will not be significant over long-term averaged runs (i.e. even though the energy partitioning changes, the total energy remains similar).

Note

currently, there is no available option for the user to set or create temperature distributions that include the radial electronic degrees of freedom with the `velocity` command, so the the user must allow for these degrees of freedom to equilibrate (i.e. equi-partitioning of energy) through time integration.

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

See the page for the `fix nvt, npt, and nph` commands for details.

2.111.5 Restrictions

This fix is part of the EFF package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

Other restriction discussed on the page for the `fix nvt, npt, and nph` commands also apply.

Note

The temperature for systems (regions or groups) with only electrons and no nuclei is 0.0 (i.e. not defined) in the current temperature calculations, a practical example would be a uniform electron gas or a very hot plasma, where electrons remain delocalized from the nuclei. This is because, even though electron virials are included in the temperature calculation, these are averaged over the nuclear degrees of freedom only. In such cases a corrective term must be added to the pressure to get the correct kinetic contribution.

2.111.6 Related commands

`fix nvt, fix nph, fix npt, fix_modify, run_style`

2.111.7 Default

The keyword defaults are tchain = 3, pchain = 3, mtk = yes, tloop = ploop = 1, nreset = 0, drag = 0.0, dilate = all, and couple = none.

(Martyna) Martyna, Tobias and Klein, J Chem Phys, 101, 4177 (1994).

(Parrinello) Parrinello and Rahman, J Appl Phys, 52, 7182 (1981).

(Tuckerman) Tuckerman, Alejandre, Lopez-Rendon, Jochim, and Martyna, J Phys A: Math Gen, 39, 5629 (2006).

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

2.112 fix nvt/uef command

2.113 fix npt/uef command

2.113.1 Syntax

```
fix ID group-ID style_name erate edot_x edot_y temp Tstart Tstop Tdamp keyword value ...
```

- ID, group-ID are documented in [fix](#) command
- style_name = *nvt/uef* or *npt/uef*
- *Tstart*, *Tstop*, and *Tdamp* are documented in the [fix npt](#) command
- *edot_x* and *edot_y* are the strain rates in the x and y directions (1/(time units))
- one or more keyword/value pairs may be appended

keyword = erate or ext or strain or temp or iso or x or y or z or tchain or pchain or tloop or ploop
→ or mtk

erate values = e_x e_y = true strain rates (required)

ext value = x or y or z or xy or yz or xz = external dimensions

sets the external dimensions used to calculate the scalar pressure

strain values = e_x e_y = initial strain

usually not needed, but may be needed to resume a run with a data file.

temp, iso, x, y, z, tchain, pchain, tloop, ploop, mtk

keywords documented by the [fix npt](#) command

2.113.2 Examples

```
fix uniax_nvt all nvt/uef temp 400 400 100 erate 0.00001 -0.000005
fix biax_nvt all nvt/uef temp 400 400 100 erate 0.000005 0.000005
fix uniax_npt all npt/uef temp 400 400 300 iso 1 1 3000 erate 0.00001 -0.000005 ext yz
fix biax_npt all npt/uef temp 400 400 100 erate -0.00001 0.000005 x 1 1 3000
```

2.113.3 Description

These fixes can be used to simulate non-equilibrium molecular dynamics (NEMD) under diagonal flow fields, including uniaxial and bi-axial flow. Simulations under continuous extensional flow may be carried out for an indefinite amount of time. It is an implementation of the boundary conditions from ([Dobson](#)), and also uses numerical lattice reduction as was proposed by ([Hunt](#)). The lattice reduction algorithm is from ([Semaev](#)). The fix is intended for simulations of homogeneous flows, and integrates the SLLOD equations of motion, originally proposed by Hoover and Ladd (see ([Evans and Morris](#))). Additional detail about this implementation can be found in ([Nicholson and Rutledge](#)).

Note that NEMD simulations of a continuously strained system can be performed using the [fix deform](#), [fix nvt/sllo](#), and [compute temp/deform](#) commands.

The applied flow field is set by the *erate* keyword. The values *edot_x* and *edot_y* correspond to the strain rates in the xx and yy directions. It is implicitly assumed that the flow field is traceless, and therefore the strain rate in the zz direction is equal to -(*edot_x* + *edot_y*).

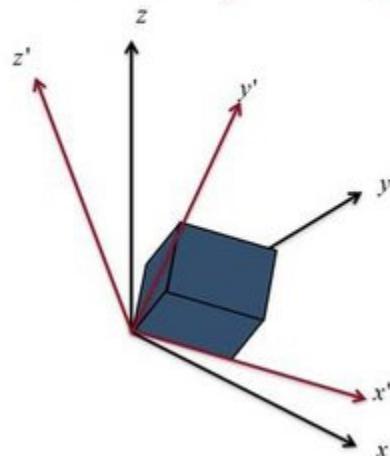
Note

Due to an instability in the SLLOD equations under extension, [fix momentum](#) should be used to regularly reset the linear momentum.

The boundary conditions require a simulation box that does not have a consistent alignment relative to the applied flow field. Since LAMMPS utilizes an upper-triangular simulation box, it is not possible to express the evolving simulation box in the same coordinate system as the flow field. These fixes keep track of two coordinate systems: the flow frame, and the upper triangular LAMMPS frame. The coordinate systems are related to each other through the QR decomposition, as is illustrated in the image below.

Flow Frame LAMMPS Frame

$$\begin{bmatrix} xx & xy & xz \\ yx & yy & yz \\ zx & zy & zz \end{bmatrix} = Q \begin{bmatrix} xx' & xy' & xz' \\ yy' & yy' & yz' \\ zz' & & \end{bmatrix}$$



During most molecular dynamics operations, the system is represented in the LAMMPS frame. Only when the positions and velocities are updated is the system rotated to the flow frame, and it is rotated back to the LAMMPS frame immediately afterwards. For this reason, all vector-valued quantities (except for the tensors from [compute pressure/uef](#) and [compute temp/uef](#)) will be computed in the LAMMPS frame. Rotationally invariant scalar quantities like the temperature and hydrostatic pressure are frame-invariant and will be computed correctly. Additionally, the system is in the LAMMPS frame during all of the output steps, and therefore trajectory files made using the dump command will be in the LAMMPS frame unless the [dump cfg/uef](#) command is used.

Temperature control is achieved with the default Nose-Hoover style thermostat documented in [fix nvt](#). When this fix is active, only the peculiar velocity of each atom is stored, defined as the velocity relative to the streaming velocity. This is in contrast to [fix nvt/sllo](#), which uses a lab-frame velocity, and removes the contribution from the streaming velocity in order to compute the temperature.

Pressure control is achieved using the default Nose-Hoover barostat documented in [fix npt](#). There are two ways to control the pressure using this fix. The first method involves using the *ext* keyword along with the *iso* pressure style. With this method, the pressure is controlled by scaling the simulation box isotropically to achieve the average pressure only in the directions specified by *ext*. For example, if the *ext* value is set to *xy*, the average pressure ($P_{xx}+P_{yy}/2$) will be controlled.

This example command will control the total hydrostatic pressure under uniaxial tension:

```
fix f1 all npt/uef temp 0.7 0.7 0.5 iso 1 1 5 erate -0.5 -0.5 ext xyz
```

This example command will control the average stress in compression directions, which would typically correspond to free surfaces under drawing with uniaxial tension:

```
fix f2 all npt/uef temp 0.7 0.7 0.5 iso 1 1 5 erate -0.5 -0.5 ext xy
```

The second method for pressure control involves setting the normal stresses using the *x*, *y*, and/or *z* keywords. When using this method, the same pressure must be specified via *Pstart* and *Pstop* for all dimensions controlled. Any choice of pressure conditions that would cause LAMMPS to compute a deviatoric stress are not permissible and will result in an error. Additionally, all dimensions with controlled stress must have the same applied strain rate. The *ext* keyword must be set to the default value (*xyz*) when using this method.

For example, the following commands will work:

```
fix f3 all npt/uef temp 0.7 0.7 0.5 x 1 1 5 y 1 1 5 erate -0.5 -0.5
fix f4 all npt/uef temp 0.7 0.7 0.5 z 1 1 5 erate 0.5 0.5
```

The following commands will not work:

```
fix f5 all npt/uef temp 0.7 0.7 0.5 x 1 1 5 z 1 1 5 erate -0.5 -0.5
fix f6 all npt/uef temp 0.7 0.7 0.5 x 1 1 5 z 2 2 5 erate 0.5 0.5
```

These fixes compute a temperature and pressure each timestep. To do this, they create their own computes of style “temp/uef” and “pressure/uef”, as if one of these two sets of commands had been issued:

```
compute fix-ID_temp group-ID temp/uef
compute fix-ID_press group-ID pressure/uef fix-ID_temp

compute fix-ID_temp all temp/uef
compute fix-ID_press all pressure/uef fix-ID_temp
```

See the [compute temp/uef](#) and [compute pressure/uef](#) commands for details. Note that the IDs of the new computes are the fix-ID + underscore + “temp” or fix_ID + underscore + “press”.

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

The fix writes the state of all the thermostat and barostat variables, as well as the cumulative strain applied, to *binary restart files*. See the [read_restart](#) command for info on how to re-specify a fix in an input script that reads a restart file, so that the operation of the fix continues in an uninterrupted fashion.

Note

It is not necessary to set the *strain* keyword when resuming a run from a restart file. Only for resuming from data files, which do not contain the cumulative applied strain, will this keyword be necessary.

These fixes can be used with the [fix_modify temp](#) and [press](#) options. The temperature and pressure computes used must be of type *temp/uef* and *pressure/uef*.

These fixes compute the same global scalar and vector quantities as [fix nvt and npt](#).

These fixes are not invoked during *energy minimization*.

2.113.5 Restrictions

These fixes are part of the UEF package. They are only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

Due to requirements of the boundary conditions, when the *strain* keyword is set to zero (or unset), the initial simulation box must be cubic and have style triclinic. If the box is initially of type ortho, use [change_box](#) before invoking the fix.

2.113.6 Related commands

[fix nvt](#), [fix npt](#), [fix nvt/slloid](#), [compute temp/uef](#), [compute pressure/uef](#), [dump cfg/uef](#)

2.113.7 Default

The default keyword values specific to these fixes are *exy* = *xyz*, *strain* = 0 0. The remaining defaults are the same as for [fix nvt or npt](#) except *tchain* = 1. The reason for this change is given in [fix nvt/slloid](#).

(Dobson) Dobson, J Chem Phys, 141, 184103 (2014).

(Hunt) Hunt, Mol Simul, 42, 347 (2016).

(Semaev) Semaev, Cryptography and Lattices, 181 (2001).

(Evans and Morriss) Evans and Morriss, Phys Rev A, 30, 1528 (1984).

(Nicholson and Rutledge) Nicholson and Rutledge, J Chem Phys, 145, 244903 (2016).

2.114 fix nonaffine/displacement command

2.114.1 Syntax

```
fix ID group nonaffine/displacement style args reference/style nstep keyword values
```

- ID, group are documented in [fix](#) command
- nonaffine/displacement = style name of this fix command
- nevery = calculate nonaffine displacement every this many timesteps
- style = *d2min* or *integrated*
 - d2min* args = cutoff args
 - cutoff = type or radius or custom
 - type args = none, cutoffs determined by atom types
 - radius args = none, cutoffs determined based on atom diameters (atom style sphere)
 - custom args = rmax, cutoff set by a constant numeric value rmax (distance units)
 - integrated args = none
 - reference/style = *fixed* or *update* or *offset*
 - fixed* = use a fixed reference frame at nstep
 - update* = update the reference frame every nstep timesteps
 - offset* = update the reference frame nstep timesteps before calculating the nonaffine displacement
 - zero or more keyword/value pairs may be appended

`z/min` values = `zmin`
`zmin` = minimum coordination number to calculate `D2min`

2.114.2 Examples

```
fix 1 all nonaffine/displacement 100 integrated update 100
fix 1 all nonaffine/displacement 1000 d2min type fixed 0
fix 1 all nonaffine/displacement 1000 d2min custom 2.0 offset 100
```

2.114.3 Description

Added in version 7Feb2024.

This fix computes different metrics of the nonaffine displacement of particles. The first metric, $d2min$ calculates the D_{\min}^2 nonaffine displacement by Falk and Langer in ([Falk](#)). For each atom, the fix computes the two tensors

$$X = \sum_{\text{neighbors}} \vec{r} (\vec{r}_0)^T$$

and

$$Y = \sum_{\text{neighbors}} \vec{r}_0 (\vec{r})^T$$

where the neighbors include all other atoms within the distance criterion set by the cutoff option, discussed below, \vec{r} is the current displacement between particles, and \vec{r}_0 is the reference displacement. A deformation gradient tensor is then calculated as $F = XY^{-1}$ from which

$$D_{\min}^2 = \sum_{\text{neighbors}} |\vec{r} - F\vec{r}_0|^2$$

and a strain tensor is calculated $E = FF^T - I$ where I is the identity tensor. This calculation is only performed on timesteps that are a multiple of `nevery` (including timestep zero). Data accessed before this occurs will simply be zeroed.

For particles with low coordination numbers, calculations of D_{\min}^2 may not be accurate. An optional minimum coordination number can be defined using the `z/min` keyword. If any particle has fewer than the specified number of particles in the cutoff distance or in contact, the above calculations will be skipped and the corresponding peratom array entries will be zero.

The `integrated` style simply integrates the velocity of particles every timestep to calculate a displacement. This style only works if used in conjunction with another fix that deforms the box and displaces atom positions such as [`fix deform`](#) with `remap x`, [`fix press/berendsen`](#), or [`fix nh`](#).

Both of these methods require defining a reference state. With the `fixed` reference style, the user picks a specific timestep `nstep` at which particle positions are saved. If peratom data is accessed from this compute prior to this timestep, it will simply be zeroed. The `update` reference style implies the reference state will be updated every `nstep` timesteps. The `offset` reference will update the reference state `nstep` timesteps before a multiple of `nevery` timesteps.

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

The reference state is saved to *binary restart files*.

None of the *fix_modify* options are relevant to this fix.

This fix computes a peratom array with 3 columns, which can be accessed by indices 1-3 using any command that uses per-atom values from a fix as input.

For the *integrated* style, the three columns are the nonaffine displacements in the x, y, and z directions. For the *d2min* style, the three columns are the calculated $\sqrt{D_{\text{min}}^2}$, the volumetric strain, and the deviatoric strain.

2.114.5 Restrictions

This compute is part of the EXTRA-FIX package. It is only enabled if LAMMPS was built with that package. See the *Build package* page for more info.

2.114.6 Related commands

none

2.114.7 Default

none

(Falk) Falk and Langer PRE, 57, 7192 (1998).

2.115 fix nph/asphere command

Accelerator Variants: *nph/asphere/omp*

2.115.1 Syntax

```
fix ID group-ID nph/asphere args keyword value ...
```

- ID, group-ID are documented in *fix* command
- nph/asphere = style name of this fix command
- additional barostat related keyword/value pairs from the *fix nph* command can be appended

2.115.2 Examples

```
fix 1 all nph/asphere iso 0.0 0.0 1000.0
fix 2 all nph/asphere x 5.0 5.0 1000.0
fix 2 all nph/asphere x 5.0 5.0 1000.0 drag 0.2
fix 2 water nph/asphere aniso 0.0 0.0 1000.0 dilate partial
```

2.115.3 Description

Perform constant NPH integration to update position, velocity, orientation, and angular velocity each timestep for aspherical or ellipsoidal particles in the group using a Nose/Hoover pressure barostat. P is pressure; H is enthalpy. This creates a system trajectory consistent with the isenthalpic ensemble.

This fix differs from the [fix nph](#) command, which assumes point particles and only updates their position and velocity.

Additional parameters affecting the barostat are specified by keywords and values documented with the [fix nph](#) command. See, for example, discussion of the *aniso*, and *dilate* keywords.

The particles in the fix group are the only ones whose velocities and positions are updated by the velocity/position update portion of the NPH integration.

Regardless of what particles are in the fix group, a global pressure is computed for all particles. Similarly, when the size of the simulation box is changed, all particles are re-scaled to new positions, unless the keyword *dilate* is specified with a value of *partial*, in which case only the particles in the fix group are re-scaled. The latter can be useful for leaving the coordinates of particles in a solid substrate unchanged and controlling the pressure of a surrounding fluid.

This fix computes a temperature and pressure each timestep. To do this, the fix creates its own computes of style “temp/asphere” and “pressure”, as if these commands had been issued:

```
compute fix-ID_temp all temp/asphere
compute fix-ID_press all pressure fix-ID_temp
```

See the [compute temp/asphere](#) and [compute pressure](#) commands for details. Note that the IDs of the new computes are the fix-ID + underscore + “temp” or fix_ID + underscore + “press”, and the group for the new computes is “all” since pressure is computed for the entire system.

Note that these are NOT the computes used by thermodynamic output (see the [thermo_style](#) command) with ID = *thermo_temp* and *thermo_press*. This means you can change the attributes of this fix’s temperature or pressure via the [compute_modify](#) command or print this temperature or pressure during thermodynamic output via the [thermo_style custom](#) command using the appropriate compute-ID. It also means that changing attributes of *thermo_temp* or *thermo_press* will have no effect on this fix.

Styles with a *gpu*, *intel*, *kk*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed on the [Accelerator packages](#) page. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the GPU, INTEL, KOKKOS, OPENMP, and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the [Build package](#) page for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the [-suffix command-line switch](#) when you invoke LAMMPS, or you can use the [suffix](#) command in your input script.

See the [Accelerator packages](#) page for more instructions on how to use the accelerated styles effectively.

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

This fix writes the state of the Nose/Hoover barostat to *binary restart files*. See the [read_restart](#) command for info on how to re-specify a fix in an input script that reads a restart file, so that the operation of the fix continues in an uninterrupted fashion.

The [fix_modify temp](#) and [press](#) options are supported by this fix. You can use them to assign a [compute](#) you have defined to this fix which will be used in its thermostating or barostating procedure. If you do this, note that the kinetic energy derived from the compute temperature should be consistent with the virial term computed using all atoms for the pressure. LAMMPS will warn you if you choose to compute temperature on a subset of atoms.

The cumulative energy change in the system imposed by this fix is included in the [thermodynamic output](#) keywords [ecouple](#) and [econserve](#). See the [thermo_style](#) doc page for details.

This fix computes the same global scalar and global vector of quantities as does the [fix nph](#) command.

This fix can ramp its target pressure over multiple runs, using the [start](#) and [stop](#) keywords of the [run](#) command. See the [run](#) command for details of how to do this.

This fix is not invoked during [energy minimization](#).

2.115.5 Restrictions

This fix is part of the ASPERE package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

This fix requires that atoms store torque and angular momentum 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.

2.115.6 Related commands

[fix nph](#), [fix nve_asphere](#), [fix nvt_asphere](#), [fix npt_asphere](#), [fix_modify](#)

2.115.7 Default

none

2.116 fix nph/body command

2.116.1 Syntax

```
fix ID group-ID nph/body args keyword value ...
```

- ID, group-ID are documented in [fix](#) command
- nph/body = style name of this fix command
- additional barostat related keyword/value pairs from the [fix nph](#) command can be appended

2.116.2 Examples

```
fix 1 all nph/body iso 0.0 0.0 1000.0
fix 2 all nph/body x 5.0 5.0 1000.0
fix 2 all nph/body x 5.0 5.0 1000.0 drag 0.2
fix 2 water nph/body aniso 0.0 0.0 1000.0 dilate partial
```

2.116.3 Description

Perform constant NPH integration to update position, velocity, orientation, and angular velocity each timestep for body particles in the group using a Nose/Hoover pressure barostat. P is pressure; H is enthalpy. This creates a system trajectory consistent with the isenthalpic ensemble.

This fix differs from the [fix nph](#) command, which assumes point particles and only updates their position and velocity.

Additional parameters affecting the barostat are specified by keywords and values documented with the [fix nph](#) command. See, for example, discussion of the *aniso*, and *dilate* keywords.

The particles in the fix group are the only ones whose velocities and positions are updated by the velocity/position update portion of the NPH integration.

Regardless of what particles are in the fix group, a global pressure is computed for all particles. Similarly, when the size of the simulation box is changed, all particles are re-scaled to new positions, unless the keyword *dilate* is specified with a value of *partial*, in which case only the particles in the fix group are re-scaled. The latter can be useful for leaving the coordinates of particles in a solid substrate unchanged and controlling the pressure of a surrounding fluid.

This fix computes a temperature and pressure each timestep. To do this, the fix creates its own computes of style “temp/body” and “pressure”, as if these commands had been issued:

```
compute fix-ID_temp all temp/body
compute fix-ID_press all pressure fix-ID_temp
```

See the [compute temp/body](#) and [compute pressure](#) commands for details. Note that the IDs of the new computes are the fix-ID + underscore + “temp” or fix_ID + underscore + “press”, and the group for the new computes is “all” since pressure is computed for the entire system.

Note that these are NOT the computes used by thermodynamic output (see the [thermo_style](#) command) with ID = *thermo_temp* and *thermo_press*. This means you can change the attributes of this fix’s temperature or pressure via the [compute_modify](#) command or print this temperature or pressure during thermodynamic output via the [thermo_style custom](#) command using the appropriate compute-ID. It also means that changing attributes of *thermo_temp* or *thermo_press* will have no effect on this fix.

Styles with a *gpu*, *intel*, *kk*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed on the [Accelerator packages](#) page. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the GPU, INTEL, KOKKOS, OPENMP, and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the [Build package](#) page for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the [-suffix command-line switch](#) when you invoke LAMMPS, or you can use the [suffix](#) command in your input script.

See the [Accelerator packages](#) page for more instructions on how to use the accelerated styles effectively.

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

This fix writes the state of the Nose/Hoover barostat to *binary restart files*. See the [read_restart](#) command for info on how to re-specify a fix in an input script that reads a restart file, so that the operation of the fix continues in an uninterrupted fashion.

The [fix_modify temp](#) and [press](#) options are supported by this fix. You can use them to assign a [compute](#) you have defined to this fix which will be used in its thermostating or barostating procedure. If you do this, note that the kinetic energy derived from the compute temperature should be consistent with the virial term computed using all atoms for the pressure. LAMMPS will warn you if you choose to compute temperature on a subset of atoms.

The cumulative energy change in the system imposed by this fix is included in the [thermodynamic output](#) keywords [ecouple](#) and [econserve](#). See the [thermo_style](#) doc page for details.

This fix computes the same global scalar and global vector of quantities as does the [fix nph](#) command.

This fix can ramp its target pressure over multiple runs, using the [start](#) and [stop](#) keywords of the [run](#) command. See the [run](#) command for details of how to do this.

This fix is not invoked during [energy minimization](#).

2.116.5 Restrictions

This fix is part of the BODY package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

This fix requires that atoms store torque and angular momentum and a quaternion as defined by the [atom_style body](#) command.

2.116.6 Related commands

[fix nph](#), [fix nve_body](#), [fix nvt_body](#), [fix npt_body](#), [fix_modify](#)

2.116.7 Default

none

2.117 fix nph/sphere command

Accelerator Variants: *nph/sphere/omp*

2.117.1 Syntax

```
fix ID group-ID nph/sphere args keyword value ...
```

- ID, group-ID are documented in [fix](#) command
- nph/sphere = style name of this fix command
- keyword = *disc*
 - disc value = none = treat particles as 2d discs, not spheres
- additional barostat related keyword/value pairs from the [fix nph](#) command can be appended

2.117.2 Examples

```
fix 1 all nph/sphere iso 0.0 0.0 1000.0
fix 2 all nph/sphere x 5.0 5.0 1000.0
fix 2 all nph/sphere x 5.0 5.0 1000.0 disc
fix 2 all nph/sphere x 5.0 5.0 1000.0 drag 0.2
fix 2 water nph/sphere aniso 0.0 0.0 1000.0 dilate partial
```

2.117.3 Description

Perform constant NPH integration to update position, velocity, and angular velocity each timestep for finite-size spherical particles in the group using a Nose/Hoover pressure barostat. P is pressure; H is enthalpy. This creates a system trajectory consistent with the isenthalpic ensemble.

This fix differs from the [fix nph](#) command, which assumes point particles and only updates their position and velocity.

If the *disc* keyword is used, then each particle is treated as a 2d disc (circle) instead of as a sphere. This is only possible for 2d simulations, as defined by the [dimension](#) keyword. The only difference between discs and spheres in this context is their moment of inertia, as used in the time integration.

Additional parameters affecting the barostat are specified by keywords and values documented with the [fix nph](#) command. See, for example, discussion of the *aniso*, and *dilate* keywords.

The particles in the fix group are the only ones whose velocities and positions are updated by the velocity/position update portion of the NPH integration.

Regardless of what particles are in the fix group, a global pressure is computed for all particles. Similarly, when the size of the simulation box is changed, all particles are re-scaled to new positions, unless the keyword *dilate* is specified with a value of *partial*, in which case only the particles in the fix group are re-scaled. The latter can be useful for leaving the coordinates of particles in a solid substrate unchanged and controlling the pressure of a surrounding fluid.

This fix computes a temperature and pressure each timestep. To do this, the fix creates its own computes of style “temp/sphere” and “pressure”, as if these commands had been issued:

```
compute fix-ID_temp all temp/sphere
compute fix-ID_press all pressure fix-ID_temp
```

See the [compute temp/sphere](#) and [compute pressure](#) commands for details. Note that the IDs of the new computes are the fix-ID + underscore + “temp” or fix_ID + underscore + “press”, and the group for the new computes is “all” since pressure is computed for the entire system.

Note that these are NOT the computes used by thermodynamic output (see the [thermo_style](#) command) with ID = *thermo_temp* and *thermo_press*. This means you can change the attributes of this fix’s temperature or pressure via the [compute_modify](#) command or print this temperature or pressure during thermodynamic output via the [thermo_style custom](#) command using the appropriate compute-ID. It also means that changing attributes of *thermo_temp* or *thermo_press* will have no effect on this fix.

Styles with a *gpu*, *intel*, *kk*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed on the [Accelerator packages](#) page. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the GPU, INTEL, KOKKOS, OPENMP, and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the [Build package](#) page for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the [-suffix command-line switch](#) when you invoke LAMMPS, or you can use the [suffix](#) command in your input script.

See the [Accelerator packages](#) page for more instructions on how to use the accelerated styles effectively.

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

This fix writes the state of the Nose/Hoover barostat to *binary restart files*. See the [read_restart](#) command for info on how to re-specify a fix in an input script that reads a restart file, so that the operation of the fix continues in an uninterrupted fashion.

The [fix_modify temp](#) and [press](#) options are supported by this fix. You can use them to assign a [compute](#) you have defined to this fix which will be used in its thermostating or barostating procedure. If you do this, note that the kinetic energy derived from the compute temperature should be consistent with the virial term computed using all atoms for the pressure. LAMMPS will warn you if you choose to compute temperature on a subset of atoms.

The cumulative energy change in the system imposed by this fix is included in the [thermodynamic output](#) keywords [ecouple](#) and [econserve](#). See the [thermo_style](#) doc page for details.

This fix computes the same global scalar and global vector of quantities as does the [fix nph](#) command.

This fix can ramp its target pressure over multiple runs, using the [start](#) and [stop](#) keywords of the [run](#) command. See the [run](#) command for details of how to do this.

This fix is not invoked during [energy minimization](#).

2.117.5 Restrictions

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

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

Use of the [disc](#) keyword is only allowed for 2d simulations, as defined by the [dimension](#) keyword.

2.117.6 Related commands

[fix nph](#), [fix nve_sphere](#), [fix nvt_sphere](#), [fix npt_sphere](#), [fix_modify](#)

2.117.7 Default

none

2.118 fix nphug command

Accelerator Variants: *nphug/omp*

2.118.1 Syntax

```
fix ID group-ID nphug keyword value ...
```

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

one or more keyword value pairs may be appended

keyword = temp or iso or aniso or tri or x or y or z or couple or tchain or pchain or mtk or tloop or ↴ploop or nreset or drag or dilate or scaleyz or scalexz or scalexy

temp values = Value1 Value2 Tdamp

Value1, Value2 = Nose-Hoover target temperatures, ignored by Hugoniostat

Tdamp = temperature damping parameter (time units)

iso or aniso or tri values = Pstart Pstop Pdamp

Pstart,Pstop = scalar external pressures, must be equal (pressure units)

Pdamp = pressure damping parameter (time units)

x or y or z or xy or yz or xz values = Pstart Pstop Pdamp

Pstart,Pstop = external stress tensor components, must be equal (pressure units)

Pdamp = stress damping parameter (time units)

couple = none or xyz or xy or yz or xz

tchain value = length of thermostat chain (1 = single thermostat)

pchain values = length of thermostat chain on barostat (0 = no thermostat)

mtk value = yes or no = add in MTK adjustment term or not

tloop value = number of sub-cycles to perform on thermostat

ploop value = number of sub-cycles to perform on barostat thermostat

nreset value = reset reference cell every this many timesteps

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

dilate value = all or partial

scaleyz value = yes or no = scale yz with lz

scalexz value = yes or no = scale xz with lz

scalexy value = yes or no = scale xy with ly

2.118.2 Examples

```
fix myhug all nphug temp 1.0 1.0 10.0 z 40.0 40.0 70.0
fix myhug all nphug temp 1.0 1.0 10.0 iso 40.0 40.0 70.0 drag 200.0 tchain 1 pchain 0
```

2.118.3 Description

This command is a variant of the Nose-Hoover [fix npt](#) fix style. It performs time integration of the Hugoniostat equations of motion developed by Ravelo et al. ([Ravelo](#)). These equations compress the system to a state with average axial stress or pressure equal to the specified target value and that satisfies the Rankine-Hugoniot (RH) jump conditions for steady shocks.

The compression can be performed either hydrostatically (using keyword *iso*, *aniso*, or *tri*) or uniaxially (using keywords *x*, *y*, or *z*). In the hydrostatic case, the cell dimensions change dynamically so that the average axial stress in all three directions converges towards the specified target value. In the uniaxial case, the chosen cell dimension changes dynamically so that the average axial stress in that direction converges towards the target value. The other two cell dimensions are kept fixed (zero lateral strain).

This leads to the following additional restrictions on the keywords:

- One and only one of the following keywords should be used: *iso*, *aniso*, *tri*, *x*, *y*, *z*
- The specified initial and final target pressures must be the same.

- The keywords *xy*, *xz*, *yz* may not be used.
- The only admissible value for the couple keyword is *xyz*, which has the same effect as keyword *iso*
- The *temp* keyword must be used to specify the time constant for kinetic energy relaxation, but initial and final target temperature values are ignored.

Essentially, a Hugoniotstat simulation is an NPT simulation in which the user-specified target temperature is replaced with a time-dependent target temperature *Tt* obtained from the following equation:

$$T_t - T = \frac{\left(\frac{1}{2}(P + P_0)(V_0 - V) + E_0 - E\right)}{N_{dof}k_B} = \Delta$$

where *T* and *Tt* are the instantaneous and target temperatures, *P* and *P0* are the instantaneous and reference pressures or axial stresses, depending on whether hydrostatic or uniaxial compression is being performed, *V* and *V0* are the instantaneous and reference volumes, *E* and *E0* are the instantaneous and reference internal energy (potential plus kinetic), *Ndof* is the number of degrees of freedom used in the definition of temperature, and *kB* is the Boltzmann constant. Δ is the negative deviation of the instantaneous temperature from the target temperature. When the system reaches a stable equilibrium, the value of Δ should fluctuate about zero.

The values of *E0*, *V0*, and *P0* are the instantaneous values at the start of the simulation. These can be overridden using the *fix_modify* keywords *e0*, *v0*, and *p0* described below.

Note

Unlike the *fix temp/berendsen* command which performs thermostating but NO time integration, this fix performs thermostating/barostatting AND time integration. Thus you should not use any other time integration fix, such as *fix nve* on atoms to which this fix is applied. Likewise, this fix should not be used on atoms that have their temperature controlled by another fix - e.g. by *fix langevin* or *fix temp/rescale* commands.

This fix computes a temperature and pressure at each timestep. To do this, the fix creates its own computes of style “temp” and “pressure”, as if one of these two sets of commands had been issued:

```
compute fix-ID_temp group-ID temp
compute fix-ID_press group-ID pressure fix-ID_temp

compute fix-ID_temp all temp
compute fix-ID_press all pressure fix-ID_temp
```

See the *compute temp* and *compute pressure* commands for details. Note that the IDs of the new computes are the *fix-ID* + underscore + “temp” or *fix_ID* + underscore + “press”. The group for the new computes is “all” since pressure is computed for the entire system.

Note that these are NOT the computes used by thermodynamic output (see the *thermo_style* command) with ID = *thermo_temp* and *thermo_press*. This means you can change the attributes of this fix’s temperature or pressure via the *compute_modify* command or print this temperature or pressure during thermodynamic output via the *thermo_style custom* command using the appropriate compute-ID. It also means that changing attributes of *thermo_temp* or *thermo_press* will have no effect on this fix.

Styles with a *gpu*, *intel*, *kk*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed on the *Accelerator packages*

page. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the GPU, INTEL, KOKKOS, OPENMP, and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the [Build package](#) page for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the [-suffix command-line switch](#) when you invoke LAMMPS, or you can use the [suffix](#) command in your input script.

See the [Accelerator packages](#) page for more instructions on how to use the accelerated styles effectively.

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

This fix writes the values of E_0 , V_0 , and P_0 , as well as the state of all the thermostat and barostat variables to [binary restart files](#). See the [read_restart](#) command for info on how to re-specify a fix in an input script that reads a restart file, so that the operation of the fix continues in an uninterrupted fashion.

The [fix_modify e0](#), [v0](#) and [p0](#) keywords can be used to define the values of E_0 , V_0 , and P_0 . Note the the values for $e0$ and $v0$ are extensive, and so must correspond to the total energy and volume of the entire system, not energy and volume per atom. If any of these quantities are not specified, then the instantaneous value in the system at the start of the simulation is used.

The [fix_modify temp](#) and [press](#) options are supported by this fix. You can use them to assign a [compute](#) you have defined to this fix which will be used in its thermostating or barostating procedure, as described above. If you do this, note that the kinetic energy derived from the compute temperature should be consistent with the virial term computed using all atoms for the pressure. LAMMPS will warn you if you choose to compute temperature on a subset of atoms.

The cumulative energy change in the system imposed by this fix is included in the [thermodynamic output](#) keywords [ecouple](#) and [econserve](#). See the [thermo_style](#) doc page for details. Note that this energy is *not* included in the definition of internal energy E when calculating the value of Delta in the above equation.

This fix computes a global scalar which can be accessed by various [output commands](#). The scalar is the same cumulative energy change due to this fix described in the previous paragraph. The scalar value calculated by this fix is “extensive”.

This fix also computes a global vector of quantities, which can be accessed by various [output commands](#). The scalar The vector values are “intensive”.

The vector stores three quantities unique to this fix (Δ , U_s , and u_p), followed by all the internal Nose/Hoover thermostat and barostat variables defined for [fix npt](#). Delta is the deviation of the temperature from the target temperature, given by the above equation. U_s and u_p are the shock and particle velocity corresponding to a steady shock calculated from the RH conditions. They have units of distance/time.

2.118.5 Restrictions

This fix style is part of the SHOCK package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

All the usual restrictions for [fix npt](#) apply, plus the additional ones mentioned above.

2.118.6 Related commands

fix msst, fix npt, fix_modify

2.118.7 Default

The keyword defaults are the same as those for *fix npt*

(Ravelo) Ravelo, Holian, Germann and Lomdahl, Phys Rev B, 70, 014103 (2004).

2.119 fix npt/asphere command

Accelerator Variants: *npt/asphere/omp*

2.119.1 Syntax

`fix ID group-ID npt/asphere keyword value ...`

- ID, group-ID are documented in *fix* command
- npt/asphere = style name of this fix command
- additional thermostat and barostat related keyword/value pairs from the *fix npt* command can be appended

2.119.2 Examples

```
fix 1 all npt/asphere temp 300.0 300.0 100.0 iso 0.0 0.0 1000.0
fix 2 all npt/asphere temp 300.0 300.0 100.0 x 5.0 5.0 1000.0
fix 2 all npt/asphere temp 300.0 300.0 100.0 x 5.0 5.0 1000.0 drag 0.2
fix 2 water npt/asphere temp 300.0 300.0 100.0 aniso 0.0 0.0 1000.0 dilate partial
```

2.119.3 Description

Perform constant NPT integration to update position, velocity, orientation, and angular velocity each timestep for aspherical or ellipsoidal particles in the group using a Nose/Hoover temperature thermostat and Nose/Hoover pressure barostat. P is pressure; T is temperature. This creates a system trajectory consistent with the isothermal-isobaric ensemble.

This fix differs from the *fix npt* command, which assumes point particles and only updates their position and velocity.

The thermostat is applied to both the translational and rotational degrees of freedom for the aspherical particles, assuming a compute is used which calculates a temperature that includes the rotational degrees of freedom (see below). The translational degrees of freedom can also have a bias velocity removed from them before thermostating takes place; see the description below.

Additional parameters affecting the thermostat and barostat are specified by keywords and values documented with the *fix npt* command. See, for example, discussion of the *temp*, *iso*, *aniso*, and *dilate* keywords.

The particles in the fix group are the only ones whose velocities and positions are updated by the velocity/position update portion of the NPT integration.

Regardless of what particles are in the fix group, a global pressure is computed for all particles. Similarly, when the size of the simulation box is changed, all particles are re-scaled to new positions, unless the keyword *dilate* is specified with a value of *partial*, in which case only the particles in the fix group are re-scaled. The latter can be useful for leaving the coordinates of particles in a solid substrate unchanged and controlling the pressure of a surrounding fluid.

This fix computes a temperature and pressure each timestep. To do this, the fix creates its own computes of style “temp/asphere” and “pressure”, as if these commands had been issued:

```
compute fix-ID_temp all temp/asphere
compute fix-ID_press all pressure fix-ID_temp
```

See the [compute temp/asphere](#) and [compute pressure](#) commands for details. Note that the IDs of the new computes are the fix-ID + underscore + “temp” or fix_ID + underscore + “press”, and the group for the new computes is “all” since pressure is computed for the entire system.

Note that these are NOT the computes used by thermodynamic output (see the [thermo_style](#) command) with ID = *thermo_temp* and *thermo_press*. This means you can change the attributes of this fix’s temperature or pressure via the [compute_modify](#) command or print this temperature or pressure during thermodynamic output via the [thermo_style custom](#) command using the appropriate compute-ID. It also means that changing attributes of *thermo_temp* or *thermo_press* will have no effect on this fix.

Like other fixes that perform thermostatting, this fix can be used with [compute commands](#) that remove a “bias” from the atom velocities. E.g. to apply the thermostat only to atoms within a spatial [region](#), or to remove the center-of-mass velocity from a group of atoms, or to remove the x-component of velocity from the calculation.

This is not done by default, but only if the [fix_modify](#) command is used to assign a temperature compute to this fix that includes such a bias term. See the doc pages for individual [compute temp commands](#) to determine which ones include a bias. In this case, the thermostat works in the following manner: bias is removed from each atom, thermostatting is performed on the remaining thermal degrees of freedom, and the bias is added back in.

Styles with a *gpu*, *intel*, *kk*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed on the [Accelerator packages](#) page. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the GPU, INTEL, KOKKOS, OPENMP, and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the [Build package](#) page for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the [-suffix command-line switch](#) when you invoke LAMMPS, or you can use the [suffix](#) command in your input script.

See the [Accelerator packages](#) page for more instructions on how to use the accelerated styles effectively.

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

This fix writes the state of the Nose/Hoover thermostat and barostat to *binary restart files*. See the [read_restart](#) command for info on how to re-specify a fix in an input script that reads a restart file, so that the operation of the fix continues in an uninterrupted fashion.

The [fix_modify temp](#) and [press](#) options are supported by this fix. You can use them to assign a [compute](#) you have defined to this fix which will be used in its thermostatting or barostatting procedure. If you do this, note that the kinetic energy derived from the compute temperature should be consistent with the virial term computed using all atoms for the pressure. LAMMPS will warn you if you choose to compute temperature on a subset of atoms.

The cumulative energy change in the system imposed by this fix is included in the *thermodynamic output* keywords *ecouple* and *econserve*. See the *thermo_style* doc page for details.

This fix computes the same global scalar and global vector of quantities as does the *fix npt* command.

This fix can ramp its target temperature and pressure over multiple runs, using the *start* and *stop* keywords of the *run* command. See the *run* command for details of how to do this.

This fix is not invoked during *energy minimization*.

2.119.5 Restrictions

This fix is part of the ASPHERE package. It is only enabled if LAMMPS was built with that package. See the *Build package* page for more info.

This fix requires that atoms store torque and angular momentum 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.

2.119.6 Related commands

fix npt, fix nve_asphere, fix nvt_asphere, fix_modify

2.119.7 Default

none

2.120 fix npt/body command

2.120.1 Syntax

```
fix ID group-ID npt/body keyword value ...
```

- ID, group-ID are documented in *fix* command
- npt/body = style name of this fix command
- additional thermostat and barostat related keyword/value pairs from the *fix npt* command can be appended

2.120.2 Examples

```
fix 1 all npt/body temp 300.0 300.0 100.0 iso 0.0 0.0 1000.0
fix 2 all npt/body temp 300.0 300.0 100.0 x 5.0 5.0 1000.0
fix 2 all npt/body temp 300.0 300.0 100.0 x 5.0 5.0 1000.0 drag 0.2
fix 2 water npt/body temp 300.0 300.0 100.0 aniso 0.0 0.0 1000.0 dilate partial
```

2.120.3 Description

Perform constant NPT integration to update position, velocity, orientation, and angular velocity each timestep for body particles in the group using a Nose/Hoover temperature thermostat and Nose/Hoover pressure barostat. P is pressure; T is temperature. This creates a system trajectory consistent with the isothermal-isobaric ensemble.

This fix differs from the [fix npt](#) command, which assumes point particles and only updates their position and velocity.

The thermostat is applied to both the translational and rotational degrees of freedom for the body particles, assuming a compute is used which calculates a temperature that includes the rotational degrees of freedom (see below). The translational degrees of freedom can also have a bias velocity removed from them before thermostating takes place; see the description below.

Additional parameters affecting the thermostat and barostat are specified by keywords and values documented with the [fix npt](#) command. See, for example, discussion of the *temp*, *iso*, *aniso*, and *dilate* keywords.

The particles in the fix group are the only ones whose velocities and positions are updated by the velocity/position update portion of the NPT integration.

Regardless of what particles are in the fix group, a global pressure is computed for all particles. Similarly, when the size of the simulation box is changed, all particles are re-scaled to new positions, unless the keyword *dilate* is specified with a value of *partial*, in which case only the particles in the fix group are re-scaled. The latter can be useful for leaving the coordinates of particles in a solid substrate unchanged and controlling the pressure of a surrounding fluid.

This fix computes a temperature and pressure each timestep. To do this, the fix creates its own computes of style “temp/body” and “pressure”, as if these commands had been issued:

```
compute fix-ID_temp all temp/body
compute fix-ID_press all pressure fix-ID_temp
```

See the [compute temp/body](#) and [compute pressure](#) commands for details. Note that the IDs of the new computes are the fix-ID + underscore + “temp” or fix_ID + underscore + “press”, and the group for the new computes is “all” since pressure is computed for the entire system.

Note that these are NOT the computes used by thermodynamic output (see the [thermo_style](#) command) with ID = *thermo_temp* and *thermo_press*. This means you can change the attributes of this fix’s temperature or pressure via the [compute_modify](#) command or print this temperature or pressure during thermodynamic output via the [thermo_style custom](#) command using the appropriate compute-ID. It also means that changing attributes of *thermo_temp* or *thermo_press* will have no effect on this fix.

Like other fixes that perform thermostating, this fix can be used with [compute commands](#) that remove a “bias” from the atom velocities. E.g. to apply the thermostat only to atoms within a spatial *region*, or to remove the center-of-mass velocity from a group of atoms, or to remove the x-component of velocity from the calculation.

This is not done by default, but only if the [fix_modify](#) command is used to assign a temperature compute to this fix that includes such a bias term. See the doc pages for individual [compute temp commands](#) to determine which ones include a bias. In this case, the thermostat works in the following manner: bias is removed from each atom, thermostating is performed on the remaining thermal degrees of freedom, and the bias is added back in.

Styles with a *gpu*, *intel*, *kk*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed on the [Accelerator packages](#) page. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the GPU, INTEL, KOKKOS, OPENMP, and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the [Build package](#) page for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the [-suffix command-line switch](#) when you invoke LAMMPS, or you can use the [suffix](#) command in your input script.

See the [Accelerator packages](#) page for more instructions on how to use the accelerated styles effectively.

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

This fix writes the state of the Nose/Hoover thermostat and barostat to [binary restart files](#). See the [read_restart](#) command for info on how to re-specify a fix in an input script that reads a restart file, so that the operation of the fix continues in an uninterrupted fashion.

The [fix_modify temp](#) and [press](#) options are supported by this fix. You can use them to assign a [compute](#) you have defined to this fix which will be used in its thermostating or barostating procedure. If you do this, note that the kinetic energy derived from the compute temperature should be consistent with the virial term computed using all atoms for the pressure. LAMMPS will warn you if you choose to compute temperature on a subset of atoms.

The cumulative energy change in the system imposed by this fix is included in the [thermodynamic output](#) keywords [ecouple](#) and [econserve](#). See the [thermo_style](#) doc page for details.

This fix computes the same global scalar and global vector of quantities as does the [fix npt](#) command.

This fix can ramp its target temperature and pressure over multiple runs, using the [start](#) and [stop](#) keywords of the [run](#) command. See the [run](#) command for details of how to do this.

This fix is not invoked during [energy minimization](#).

2.120.5 Restrictions

This fix is part of the BODY package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

This fix requires that atoms store torque and angular momentum and a quaternion as defined by the [atom_style body](#) command.

2.120.6 Related commands

[fix npt](#), [fix nve_body](#), [fix nvt_body](#), [fix_modify](#)

2.120.7 Default

none

2.121 fix npt/cauchy command

2.121.1 Syntax

```
fix ID group-ID style_name keyword value ...
```

- ID, group-ID are documented in [fix](#) command
- style_name = *npt/cauchy*
- one or more keyword/value pairs may be appended

- keyword = *temp* or *iso* or *aniso* or *tri* or *x* or *y* or *z* or *xy* or *yz* or *xz* or *couple* or *tchain* or *pchain* or *mtk* or *tloop* or *ploop* or *nreset* or *drag* or *dilate* or *scalexy* or *scaleyz* or *scalexz* or *flip* or *alpha* or *continue* or *fixedpoint*

temp values = Tstart Tstop Tdamp

Tstart,Tstop = external temperature at start/end of run

Tdamp = temperature damping parameter (time units)

iso or aniso or tri values = Pstart Pstop Pdamp

Pstart,Pstop = scalar external pressure at start/end of run (pressure units)

Pdamp = pressure damping parameter (time units)

x or y or z or xy or yz or xz values = Pstart Pstop Pdamp

Pstart,Pstop = external stress tensor component at start/end of run (pressure units)

Pdamp = stress damping parameter (time units)

couple = none or xyz or xy or yz or xz

tchain value = N

N = length of thermostat chain (1 = single thermostat)

pchain values = N

N length of thermostat chain on barostat (0 = no thermostat)

mtk value = yes or no = add in MTK adjustment term or not

tloop value = M

M = number of sub-cycles to perform on thermostat

ploop value = M

M = number of sub-cycles to perform on barostat thermostat

nreset value = reset reference cell every this many timesteps

drag value = Df

Df = drag factor added to barostat/thermostat (0.0 = no drag)

dilate value = dilate-group-ID

dilate-group-ID = only dilate atoms in this group due to barostat volume changes

scalexy value = yes or no = scale xy with ly

scaleyz value = yes or no = scale yz with lz

scalexz value = yes or no = scale xz with lz

flip value = yes or no = allow or disallow box flips when it becomes highly skewed

alpha value = strength of Cauchy stress control parameter

continue value = yes or no = whether of not to continue from a previous run

fixedpoint values = x y z

x,y,z = perform barostat dilation/contraction around this point (distance units)

2.121.2 Examples

```
fix 1 water npt/cauchy temp 300.0 300.0 100.0 iso 0.0 0.0 1000.0 alpha 0.001
```

2.121.3 Description

This command performs time integration on Nose-Hoover style non-Hamiltonian equations of motion which are designed to generate positions and velocities sampled from the isothermal-isobaric (npt) ensembles. This updates the position and velocity for atoms in the group each timestep and the box dimensions.

The thermostating and barostatting is achieved by adding some dynamic variables which are coupled to the particle velocities (thermostating) and simulation domain dimensions (barostatting). In addition to basic thermostating and barostatting, this fix can also create a chain of thermostats coupled to the particle thermostat, and another chain of thermostats coupled to the barostat variables. The barostat can be coupled to the overall box volume, or to individual dimensions, including the *xy*, *xz* and *yz* tilt dimensions. The external pressure of the barostat can be specified as either a scalar pressure (isobaric ensemble) or as components of a symmetric stress tensor (constant stress ensemble). When

used correctly, the time-averaged temperature and stress tensor of the particles will match the target values specified by *Tstart/Tstop* and *Pstart/Pstop*.

The equations of motion used are those of Shinoda et al in ([Shinoda](#)), which combine the hydrostatic equations of Martyna, Tobias and Klein in ([Martyna](#)) with the strain energy proposed by Parrinello and Rahman in ([Parrinello](#)). The time integration schemes closely follow the time-reversible measure-preserving Verlet and rRESPA integrators derived by Tuckerman et al in ([Tuckerman](#)).

The thermostat parameters are specified using the *temp* keyword. Other thermostat-related keywords are *tchain*, *tloop* and *drag*, which are discussed below.

The thermostat is applied to only the translational degrees of freedom for the particles. The translational degrees of freedom can also have a bias velocity removed before thermostating takes place; see the description below. The desired temperature at each timestep is a ramped value during the run from *Tstart* to *Tstop*. The *Tdamp* parameter is specified in time units and determines how rapidly the temperature is relaxed. For example, a value of 10.0 means to relax the temperature in a timespan of (roughly) 10 time units (e.g. τ or fs or ps - see the [units](#) command). The atoms in the fix group are the only ones whose velocities and positions are updated by the velocity/position update portion of the integration.

Note

A Nose-Hoover thermostat will not work well for arbitrary values of *Tdamp*. If *Tdamp* is too small, the temperature can fluctuate wildly; if it is too large, the temperature will take a very long time to equilibrate. A good choice for many models is a *Tdamp* of around 100 timesteps. Note that this is NOT the same as 100 time units for most [units](#) settings.

The barostat parameters are specified using one or more of the *iso*, *aniso*, *tri*, *x*, *y*, *z*, *xy*, *xz*, *yz*, and *couple* keywords. These keywords give you the ability to specify all 6 components of an external stress tensor, and to couple various of these components together so that the dimensions they represent are varied together during a constant-pressure simulation.

Other barostat-related keywords are *pchain*, *mtk*, *ploop*, *nreset*, *drag*, and *dilate*, which are discussed below.

Orthogonal simulation boxes have 3 adjustable dimensions (x,y,z). Triclinic (non-orthogonal) simulation boxes have 6 adjustable dimensions (x,y,z,xy,xz,yz). The *create_box*, *read data*, and *read_restart* commands specify whether the simulation box is orthogonal or non-orthogonal (triclinic) and explain the meaning of the xy,xz,yz tilt factors.

The target pressures for each of the 6 components of the stress tensor can be specified independently via the *x*, *y*, *z*, *xy*, *xz*, *yz* keywords, which correspond to the 6 simulation box dimensions. For each component, the external pressure or tensor component at each timestep is a ramped value during the run from *Pstart* to *Pstop*. If a target pressure is specified for a component, then the corresponding box dimension will change during a simulation. For example, if the *y* keyword is used, the *y*-box length will change. If the *xy* keyword is used, the *xy* tilt factor will change. A box dimension will not change if that component is not specified, although you have the option to change that dimension via the [fix deform](#) command.

Note that in order to use the *xy*, *xz*, or *yz* keywords, the simulation box must be triclinic, even if its initial tilt factors are 0.0.

For all barostat keywords, the *Pdamp* parameter operates like the *Tdamp* parameter, determining the time scale on which pressure is relaxed. For example, a value of 10.0 means to relax the pressure in a timespan of (roughly) 10 time units (e.g. τ or fs or ps - see the [units](#) command).

Note

A Nose-Hoover barostat will not work well for arbitrary values of *Pdamp*. If *Pdamp* is too small, the pressure and volume can fluctuate wildly; if it is too large, the pressure will take a very long time to equilibrate. A good choice for many models is a *Pdamp* of around 1000 timesteps. However, note that *Pdamp* is specified in time units, and that timesteps are NOT the same as time units for most *units* settings.

Regardless of what atoms are in the fix group (the only atoms which are time integrated), a global pressure or stress tensor is computed for all atoms. Similarly, when the size of the simulation box is changed, all atoms are re-scaled to new positions, unless the keyword *dilate* is specified with a *dilate-group-ID* for a group that represents a subset of the atoms. This can be useful, for example, to leave the coordinates of atoms in a solid substrate unchanged and controlling the pressure of a surrounding fluid. This option should be used with care, since it can be unphysical to dilate some atoms and not others, because it can introduce large, instantaneous displacements between a pair of atoms (one dilated, one not) that are far from the dilation origin. Also note that for atoms not in the fix group, a separate time integration fix like *fix nve* or *fix nvt* can be used on them, independent of whether they are dilated or not.

The *couple* keyword allows two or three of the diagonal components of the pressure tensor to be “coupled” together. The value specified with the keyword determines which are coupled. For example, *xz* means the *Pxx* and *Pzz* components of the stress tensor are coupled. *Xyz* means all 3 diagonal components are coupled. Coupling means two things: the instantaneous stress will be computed as an average of the corresponding diagonal components, and the coupled box dimensions will be changed together in lockstep, meaning coupled dimensions will be dilated or contracted by the same percentage every timestep. The *Pstart*, *Pstop*, *Pdamp* parameters for any coupled dimensions must be identical. *Couple xyz* can be used for a 2d simulation; the *z* dimension is simply ignored.

The *iso*, *aniso*, and *tri* keywords are simply shortcuts that are equivalent to specifying several other keywords together. The keyword *iso* means couple all 3 diagonal components together when pressure is computed (hydrostatic pressure), and dilate/contract the dimensions together. Using “*iso Pstart Pstop Pdamp*” is the same as specifying these 4 keywords:

```
x Pstart Pstop Pdamp
y Pstart Pstop Pdamp
z Pstart Pstop Pdamp
couple xyz
```

The keyword *aniso* means *x*, *y*, and *z* dimensions are controlled independently using the *Pxx*, *Pyy*, and *Pzz* components of the stress tensor as the driving forces, and the specified scalar external pressure. Using “*aniso Pstart Pstop Pdamp*” is the same as specifying these 4 keywords:

```
x Pstart Pstop Pdamp
y Pstart Pstop Pdamp
z Pstart Pstop Pdamp
couple none
```

The keyword *tri* means *x*, *y*, *z*, *xy*, *xz*, and *yz* dimensions are controlled independently using their individual stress components as the driving forces, and the specified scalar pressure as the external normal stress. Using “*tri Pstart Pstop Pdamp*” is the same as specifying these 7 keywords:

```
x Pstart Pstop Pdamp
y Pstart Pstop Pdamp
z Pstart Pstop Pdamp
xy 0.0 0.0 Pdamp
```

(continues on next page)

(continued from previous page)

```
yz 0.0 0.0 Pdamp
xz 0.0 0.0 Pdamp
couple none
```

In some cases (e.g. for solids) the pressure (volume) and/or temperature of the system can oscillate undesirably when a Nose/Hoover barostat and thermostat is applied. The optional *drag* keyword will damp these oscillations, although it alters the Nose/Hoover equations. A value of 0.0 (no drag) leaves the Nose/Hoover formalism unchanged. A non-zero value adds a drag term; the larger the value specified, the greater the damping effect. Performing a short run and monitoring the pressure and temperature is the best way to determine if the drag term is working. Typically a value between 0.2 to 2.0 is sufficient to damp oscillations after a few periods. Note that use of the *drag* keyword will interfere with energy conservation and will also change the distribution of positions and velocities so that they do not correspond to the nominal NVT, NPT, or NPH ensembles.

An alternative way to control initial oscillations is to use chain thermostats. The keyword *tchain* determines the number of thermostats in the particle thermostat. A value of 1 corresponds to the original Nose-Hoover thermostat. The keyword *pchain* specifies the number of thermostats in the chain thermostatting the barostat degrees of freedom. A value of 0 corresponds to no thermostatting of the barostat variables.

The *mtk* keyword controls whether or not the correction terms due to Martyna, Tuckerman, and Klein are included in the equations of motion ([Martyna](#)). Specifying *no* reproduces the original Hoover barostat, whose volume probability distribution function differs from the true NPT and NPH ensembles by a factor of $1/V$. Hence using *yes* is more correct, but in many cases the difference is negligible.

The keyword *tloop* can be used to improve the accuracy of integration scheme at little extra cost. The initial and final updates of the thermostat variables are broken up into *tloop* sub-steps, each of length $dt/tloop$. This corresponds to using a first-order Suzuki-Yoshida scheme ([Tuckerman](#)). The keyword *ploop* does the same thing for the barostat thermostat.

The keyword *nreset* controls how often the reference dimensions used to define the strain energy are reset. If this keyword is not used, or is given a value of zero, then the reference dimensions are set to those of the initial simulation domain and are never changed. If the simulation domain changes significantly during the simulation, then the final average pressure tensor will differ significantly from the specified values of the external stress tensor. A value of *nstep* means that every *nstep* timesteps, the reference dimensions are set to those of the current simulation domain.

The *scaleyz*, *scalexz*, and *scalexy* keywords control whether or not the corresponding tilt factors are scaled with the associated box dimensions when barostatting triclinic periodic cells. The default values *yes* will turn on scaling, which corresponds to adjusting the linear dimensions of the cell while preserving its shape. Choosing *no* ensures that the tilt factors are not scaled with the box dimensions. See below for restrictions and default values in different situations. In older versions of LAMMPS, scaling of tilt factors was not performed. The old behavior can be recovered by setting all three scale keywords to *no*.

The *flip* keyword allows the tilt factors for a triclinic box to exceed half the distance of the parallel box length, as discussed below. 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 applied stress induces large deformations (e.g. in a liquid), this means the box shape can tilt dramatically and LAMMPS will run less efficiently, due to the large volume of communication needed to acquire ghost atoms around a processor's irregular-shaped subdomain. For extreme values of tilt, LAMMPS may also lose atoms and generate an error.

The *fixedpoint* keyword specifies the fixed point for barostat volume changes. 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 trajectories are unaffected by the chosen value, except for a time-dependent constant translation of positions.

Note

Using a barostat coupled to tilt dimensions xy , xz , yz can sometimes result in arbitrarily large values of the tilt dimensions, i.e. a dramatically deformed simulation box. LAMMPS allows the tilt factors to grow a small amount beyond the normal limit of half the box length (0.6 times the box length), and then performs a box “flip” to an equivalent periodic cell. See the discussion of the *flip* keyword above, to allow this bound to be exceeded, if desired.

The flip operation is described in more detail in the page for [fix deform](#). Both the barostat dynamics and the atom trajectories are unaffected by this operation. However, if a tilt factor is incremented by a large amount (1.5 times the box length) on a single timestep, LAMMPS can not accommodate this event and will terminate the simulation with an error. This error typically indicates that there is something badly wrong with how the simulation was constructed, such as specifying values of P_{start} that are too far from the current stress value, or specifying a timestep that is too large. Triclinic barostatting should be used with care. This also is true for other barostat styles, although they tend to be more forgiving of insults. In particular, it is important to recognize that equilibrium liquids can not support a shear stress and that equilibrium solids can not support shear stresses that exceed the yield stress.

One exception to this rule is if the first 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.

Note

Unlike the [fix temp/berendsen](#) command which performs thermostatting but NO time integration, this fix performs thermostatting/barostatting AND time integration. Thus you should not use any other time integration fix, such as [fix nve](#) on atoms to which this fix is applied. Likewise, [fix npt/cauchy](#) should not normally be used on atoms that also have their temperature controlled by another fix - e.g. by [fix langevin](#) or [fix temp/rescale](#) commands.

See the [Howto thermostat](#) and [Howto barostat](#) doc pages for a discussion of different ways to compute temperature and perform thermostatting and barostatting.

This fix compute a temperature and pressure each timestep. To do this, the fix creates its own computes of style “temp” and “pressure”, as if one of these sets of commands had been issued:

```
compute fix-ID_temp all temp
compute fix-ID_press all pressure fix-ID_temp
```

The group for both the new temperature and pressure compute is “all” since pressure is computed for the entire system. 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”.

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 these fix’s temperature or pressure via the [compute_modify](#) command. Or you can print this temperature or pressure during thermodynamic output via the [thermo_style custom](#) command using the appropriate compute-ID. It also means that changing attributes of *thermo_temp* or *thermo_press* will have no effect on this fix.

Like other fixes that perform thermostatting, this fix can be used with [compute commands](#) that remove a “bias” from the atom velocities. E.g. to apply the thermostat only to atoms within a spatial [region](#), or to remove the center-of-mass velocity from a group of atoms, or to remove the x-component of velocity from the calculation.

This is not done by default, but only if the [fix_modify](#) command is used to assign a temperature compute to this fix that includes such a bias term. See the doc pages for individual [compute temp commands](#) to determine which ones include

a bias. In this case, the thermostat works in the following manner: bias is removed from each atom, thermostatting is performed on the remaining thermal degrees of freedom, and the bias is added back in.

This fix can be used with either the *verlet* or *respa* *integrators*. When using this fix with *respa*, LAMMPS uses an integrator constructed according to the following factorization of the Liouville propagator (for two rRESPA levels):

$$\begin{aligned} \exp(iL\Delta t) = & \hat{E} \exp\left(iL_{T,\text{baro}} \frac{\Delta t}{2}\right) \exp\left(iL_{T,\text{part}} \frac{\Delta t}{2}\right) \exp\left(iL_{\epsilon,2} \frac{\Delta t}{2}\right) \exp\left(iL_2^{(2)} \frac{\Delta t}{2}\right) \\ & \times \left[\exp\left(iL_2^{(1)} \frac{\Delta t}{2n}\right) \exp\left(iL_{\epsilon,1} \frac{\Delta t}{2n}\right) \exp\left(iL_1 \frac{\Delta t}{n}\right) \exp\left(iL_{\epsilon,1} \frac{\Delta t}{2n}\right) \exp\left(iL_2^{(1)} \frac{\Delta t}{2n}\right) \right]^n \\ & \times \exp\left(iL_2^{(2)} \frac{\Delta t}{2}\right) \exp\left(iL_{\epsilon,2} \frac{\Delta t}{2}\right) \exp\left(iL_{T,\text{part}} \frac{\Delta t}{2}\right) \exp\left(iL_{T,\text{baro}} \frac{\Delta t}{2}\right) \\ & + \mathcal{O}(\Delta t^3) \end{aligned}$$

This factorization differs somewhat from that of Tuckerman et al, in that the barostat is only updated at the outermost rRESPA level, whereas Tuckerman's factorization requires splitting the pressure into pieces corresponding to the forces computed at each rRESPA level. In theory, the latter method will exhibit better numerical stability. In practice, because Pdamp is normally chosen to be a large multiple of the outermost rRESPA timestep, the barostat dynamics are not the limiting factor for numerical stability. Both factorizations are time-reversible and can be shown to preserve the phase space measure of the underlying non-Hamiltonian equations of motion.

Note

Under NPT dynamics, for a system with zero initial total linear momentum, the total momentum fluctuates close to zero. It may occasionally undergo brief excursions to non-negligible values, before returning close to zero. Over long simulations, this has the effect of causing the center-of-mass to undergo a slow random walk. This can be mitigated by resetting the momentum at infrequent intervals using the [fix momentum](#) command.

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

This fix writes the state of all the thermostat and barostat variables to *binary restart files*. See the [read_restart](#) command for info on how to re-specify a fix in an input script that reads a restart file, so that the operation of the fix continues in an uninterrupted fashion.

The [fix_modify temp](#) and [press](#) options are supported by this fix. You can use them to assign a [compute](#) you have defined to this fix which will be used in its thermostatting or barostatting procedure, as described above. If you do this, note that the kinetic energy derived from the compute temperature should be consistent with the virial term computed using all atoms for the pressure. LAMMPS will warn you if you choose to compute temperature on a subset of atoms.

Note

If both the *temp* and *press* keywords are used in a single thermo_modify command (or in two separate commands), then the order in which the keywords are specified is important. Note that a [pressure compute](#) defines its own temperature compute as an argument when it is specified. The *temp* keyword will override this (for the pressure compute being used by fix npt), but only if the *temp* keyword comes after the *press* keyword. If the *temp* keyword comes before the *press* keyword, then the new pressure compute specified by the *press* keyword will be unaffected by the *temp* setting.

The cumulative energy change in the system imposed by this fix, due to thermostating and/or barostatting, is included in the [thermodynamic output](#) keywords `ecouple` and `econserve`. See the [thermo_style](#) page for details.

This fix computes a global scalar which can be accessed by various [output commands](#). The scalar is the same cumulative energy change due to this fix described in the previous paragraph. The scalar value calculated by this fix is “extensive”.

This fix also computes a global vector of quantities, which can be accessed by various [output commands](#). The vector values are “intensive”.

The vector stores internal Nose/Hoover thermostat and barostat variables. The number and meaning of the vector values depends on which fix is used and the settings for keywords `tchain` and `pchain`, which specify the number of Nose/Hoover chains for the thermostat and barostat. If no thermostating is done, then `tchain` is 0. If no barostatting is done, then `pchain` is 0. In the following list, “ndof” is 0, 1, 3, or 6, and is the number of degrees of freedom in the barostat. Its value is 0 if no barostat is used, else its value is 6 if any off-diagonal stress tensor component is barostatted, else its value is 1 if `couple xyz` is used or `couple xy` for a 2d simulation, otherwise its value is 3.

The order of values in the global vector and their meaning is as follows. The notation means there are `tchain` values for `eta`, followed by `tchain` for `eta_dot`, followed by `ndof` for `omega`, etc:

- `eta[tchain]` = particle thermostat displacements (unitless)
- `eta_dot[tchain]` = particle thermostat velocities (1/time units)
- `omega[ndof]` = barostat displacements (unitless)
- `omega_dot[ndof]` = barostat velocities (1/time units)
- `etap[pchain]` = barostat thermostat displacements (unitless)
- `etap_dot[pchain]` = barostat thermostat velocities (1/time units)
- `PE_eta[tchain]` = potential energy of each particle thermostat displacement (energy units)
- `KE_eta_dot[tchain]` = kinetic energy of each particle thermostat velocity (energy units)
- `PE_omega[ndof]` = potential energy of each barostat displacement (energy units)
- `KE_omega_dot[ndof]` = kinetic energy of each barostat velocity (energy units)
- `PE_etap[pchain]` = potential energy of each barostat thermostat displacement (energy units)
- `KE_etap_dot[pchain]` = kinetic energy of each barostat thermostat velocity (energy units)
- `PE_strain[1]` = scalar strain energy (energy units)

This fix can ramp its external temperature and pressure over multiple runs, using the `start` and `stop` keywords of the [run](#) command. See the [run](#) command for details of how to do this.

This fix is not invoked during [energy minimization](#).

2.121.5 Restrictions

This fix is part of the EXTRA-FIX package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

X, y, z cannot be barostatted if the associated dimension is not periodic. Xy, xz , and yz can only be barostatted if the simulation domain is triclinic and the second dimension in the keyword (y dimension in xy) is periodic. Z, xz , and yz , cannot be barostatted for 2D simulations. The `create_box`, `read data`, and `read_restart` commands specify whether the simulation box is orthogonal or non-orthogonal (triclinic) and explain the meaning of the xy, xz, yz tilt factors.

For the `temp` keyword, the final `Tstop` cannot be 0.0 since it would make the external $T = 0.0$ at some timestep during the simulation which is not allowed in the Nose/Hoover formulation.

The `scaleyz yes` and `scalexz yes` keyword/value pairs can not be used for 2D simulations. `scaleyz yes`, `scalexz yes`, and `scalexy yes` options can only be used if the second dimension in the keyword is periodic, and if the tilt factor is not coupled to the barostat via keywords `tri`, `yz`, `xz`, and `xy`.

The `alpha` keyword modifies the barostat as per Miller et al. (Miller)_ "#nc-Miller" so that the Cauchy stress is controlled. `alpha` is the non-dimensional parameter, typically set to 0.001 or 0.01 that determines how aggressively the algorithm drives the system towards the set Cauchy stresses. Larger values of `alpha` will modify the system more quickly, but can lead to instabilities. Smaller values will lead to longer convergence time. Since `alpha` also influences how much the stress fluctuations deviate from the equilibrium fluctuations, it should be set as small as possible.

A `continue` value of `yes` indicates that the fix is subsequent to a previous run with the `npt/cauchy` fix, and the intention is to continue from the converged stress state at the end of the previous run. This may be required, for example, when implementing a multi-step loading/unloading sequence over several fixes.

Setting `alpha` to zero is not permitted. To "turn off" the `cauchystat` control and thus restore the equilibrium stress fluctuations, two subsequent fixes should be used. In the first, `fix npt/cauchy` is used and the simulation box equilibrates to the correct shape for the desired stresses. In the second, `fix npt` is used instead which uses the original Parrinello-Rahman algorithm, but now with the corrected simulation box shape from using `fix npt/cauchy`.

This fix can be used with dynamic groups as defined by the `group` command. Likewise it can be used with groups to which atoms are added or deleted over time, e.g. a deposition simulation. However, the conservation properties of the thermostat and barostat are defined for systems with a static set of atoms. You may observe odd behavior if the atoms in a group vary dramatically over time or the atom count becomes very small.

2.121.6 Related commands

`fix nve`, `fix_modify`, `run_style`

2.121.7 Default

The keyword defaults are `tchain = 3`, `pchain = 3`, `mtk = yes`, `tloop = ploop = 1`, `nreset = 0`, `drag = 0.0`, `dilate = all`, `couple = none`, `cauchystat = no`, `scaleyz = scalexz = scalexy = yes` if periodic in second dimension and not coupled to barostat, otherwise `no`.

(**Martyna**) Martyna, Tobias and Klein, J Chem Phys, 101, 4177 (1994).

(**Parrinello**) Parrinello and Rahman, J Appl Phys, 52, 7182 (1981).

(**Tuckerman**) Tuckerman, Alejandre, Lopez-Rendon, Jochim, and Martyna, J Phys A: Math Gen, 39, 5629 (2006).

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

(**Miller**) Miller, Tadmor, Gibson, Bernstein and Pavia, J Chem Phys, 144, 184107 (2016).

2.122 fix npt/sphere command

Accelerator Variants: `npt/sphere/omp`

2.122.1 Syntax

```
fix ID group-ID npt/sphere keyword value ...
```

- ID, group-ID are documented in [fix](#) command npt/sphere = style name of this fix command zero or more keyword/value pairs may be appended
 - keyword = *disc*
- disc value = none = treat particles as 2d discs, not spheres
- NOTE: additional thermostat and barostat and dipole related keyword/value pairs from the [fix npt](#) command can be appended

2.122.2 Examples

```
fix 1 all npt/sphere temp 300.0 300.0 100.0 iso 0.0 0.0 1000.0
fix 2 all npt/sphere temp 300.0 300.0 100.0 x 5.0 5.0 1000.0
fix 2 all npt/sphere temp 300.0 300.0 100.0 x 5.0 5.0 1000.0 disc
fix 2 all npt/sphere temp 300.0 300.0 100.0 x 5.0 5.0 1000.0 drag 0.2
fix 2 all npt/sphere temp 300.0 300.0 100.0 x 5.0 5.0 1000.0 update dipole
fix 2 water npt/sphere temp 300.0 300.0 100.0 aniso 0.0 0.0 1000.0 dilate partial
```

2.122.3 Description

Perform constant NPT integration to update position, velocity, and angular velocity each timestep for finite-size spherical particles in the group using a Nose/Hoover temperature thermostat and Nose/Hoover pressure barostat. P is pressure; T is temperature. This creates a system trajectory consistent with the isothermal-isobaric ensemble.

This fix differs from the [fix npt](#) command, which assumes point particles and only updates their position and velocity.

The thermostat is applied to both the translational and rotational degrees of freedom for the spherical particles, assuming a compute is used which calculates a temperature that includes the rotational degrees of freedom (see below). The translational degrees of freedom can also have a bias velocity removed from them before thermostating takes place; see the description below.

If the *disc* keyword is used, then each particle is treated as a 2d disc (circle) instead of as a sphere. This is only possible for 2d simulations, as defined by the [dimension](#) keyword. The only difference between discs and spheres in this context is their moment of inertia, as used in the time integration.

Additional parameters affecting the thermostat and barostat are specified by keywords and values documented with the [fix npt](#) command. See, for example, discussion of the *temp*, *iso*, *aniso*, and *dilate* keywords.

The particles in the fix group are the only ones whose velocities and positions are updated by the velocity/position update portion of the NPT integration.

Regardless of what particles are in the fix group, a global pressure is computed for all particles. Similarly, when the size of the simulation box is changed, all particles are re-scaled to new positions, unless the keyword *dilate* is specified with a value of *partial*, in which case only the particles in the fix group are re-scaled. The latter can be useful for leaving the coordinates of particles in a solid substrate unchanged and controlling the pressure of a surrounding fluid.

This fix computes a temperature and pressure each timestep. To do this, the fix creates its own computes of style “temp/sphere” and “pressure”, as if these commands had been issued:

```
compute fix-ID_temp all temp/sphere
compute fix-ID_press all pressure fix-ID_temp
```

See the [compute temp/sphere](#) and [compute pressure](#) commands for details. Note that the IDs of the new computes are the fix-ID + underscore + “temp” or fix_ID + underscore + “press”, and the group for the new computes is “all” since pressure is computed for the entire system.

Note that these are NOT the computes used by thermodynamic output (see the [thermo_style](#) command) with ID = *thermo_temp* and *thermo_press*. This means you can change the attributes of this fix’s temperature or pressure via the [compute_modify](#) command or print this temperature or pressure during thermodynamic output via the [thermo_style custom](#) command using the appropriate compute-ID. It also means that changing attributes of *thermo_temp* or *thermo_press* will have no effect on this fix.

Like other fixes that perform thermostating, this fix can be used with [compute commands](#) that remove a “bias” from the atom velocities. E.g. to apply the thermostat only to atoms within a spatial [region](#), or to remove the center-of-mass velocity from a group of atoms, or to remove the x-component of velocity from the calculation.

This is not done by default, but only if the [fix_modify](#) command is used to assign a temperature compute to this fix that includes such a bias term. See the doc pages for individual [compute temp commands](#) to determine which ones include a bias. In this case, the thermostat works in the following manner: bias is removed from each atom, thermostating is performed on the remaining thermal degrees of freedom, and the bias is added back in.

Styles with a *gpu*, *intel*, *kk*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed on the [Accelerator packages](#) page. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the GPU, INTEL, KOKKOS, OPENMP, and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the [Build package](#) page for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the [-suffix command-line switch](#) when you invoke LAMMPS, or you can use the [suffix](#) command in your input script.

See the [Accelerator packages](#) page for more instructions on how to use the accelerated styles effectively.

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

This fix writes the state of the Nose/Hoover thermostat and barostat to [binary restart files](#). See the [read_restart](#) command for info on how to re-specify a fix in an input script that reads a restart file, so that the operation of the fix continues in an uninterrupted fashion.

The [fix_modify temp](#) and [press](#) options are supported by this fix. You can use them to assign a [compute](#) you have defined to this fix which will be used in its thermostating or barostating procedure. If you do this, note that the kinetic energy derived from the compute temperature should be consistent with the virial term computed using all atoms for the pressure. LAMMPS will warn you if you choose to compute temperature on a subset of atoms.

The cumulative energy change in the system imposed by this fix is included in the [thermodynamic output](#) keywords [ecouple](#) and [econserve](#). See the [thermo_style](#) doc page for details.

This fix computes the same global scalar and global vector of quantities as does the [fix npt](#) command.

This fix can ramp its target temperature and pressure over multiple runs, using the [start](#) and [stop](#) keywords of the [run](#) command. See the [run](#) command for details of how to do this.

This fix is not invoked during [energy minimization](#).

2.122.5 Restrictions

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

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

Use of the *disc* keyword is only allowed for 2d simulations, as defined by the [dimension](#) keyword.

2.122.6 Related commands

```
fix npt,fix nve_sphere,  
fix nvt_sphere,fix npt_asphere,fix_modify
```

2.122.7 Default

none

2.123 fix numdiff command

2.123.1 Syntax

```
fix ID group-ID numdiff Nevery delta
```

- ID, group-ID are documented in [fix](#) command
- numdiff = style name of this fix command
- Nevery = calculate force by finite difference every this many timesteps
- delta = size of atom displacements (distance units)

2.123.2 Examples

```
fix 1 all numdiff 10 1e-6  
fix 1 movegroup numdiff 100 0.01
```

2.123.3 Description

Calculate forces through finite difference calculations of energy versus position. These forces can be compared to analytic forces computed by pair styles, bond styles, etc. This can be useful for debugging or other purposes.

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.

This fix performs a loop over all atoms in the group. For each atom and each component of force it adds *delta* to the position, and computes the new energy of the entire system. It then subtracts *delta* from the original position and again computes the new energy of the system. It then restores the original position. That component of force is calculated as the difference in energy divided by two times *delta*.

Note

It is important to choose a suitable value for delta, the magnitude of atom displacements that are used to generate finite difference approximations to the exact forces. For typical systems, a value in the range of 1 part in 1e4 to 1e5 of the typical separation distance between atoms in the liquid or solid state will be sufficient. However, the best value will depend on a multitude of factors including the stiffness of the interatomic potential, the thermodynamic state of the material being probed, and so on. The only way to be sure that you have made a good choice is to do a sensitivity study on a representative atomic configuration, sweeping over a wide range of values of delta. If delta is too small, the output forces will vary erratically due to truncation effects. If delta is increased beyond a certain point, the output forces will start to vary smoothly with delta, due to growing contributions from higher order derivatives. In between these two limits, the numerical force values should be largely independent of delta.

Note

The cost of each energy evaluation is essentially the cost of an MD timestep. Thus invoking this fix once for a 3d system has a cost of $6N$ timesteps, where N is the total number of atoms in the system. So this fix can be very expensive to use for large systems. One expedient alternative is to define the fix for a group containing only a few atoms.

The *Nevery* argument specifies on what timesteps the force will be used calculated by finite difference.

The *delta* argument specifies the size of the displacement each atom will undergo.

2.123.4 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*, which stores the components of the force on each atom as calculated by finite difference. The per-atom values can only be accessed on timesteps that are multiples of *Nevery* since that is when the finite difference forces are calculated. See the examples in *examples/numdiff* directory to see how this fix can be used to directly compare with the analytic forces computed by LAMMPS.

The array values calculated by this compute will be in force *units*.

No parameter of this fix can be used with the *start/stop* keywords of the *run* command. This fix is invoked during *energy minimization*.

2.123.5 Restrictions

This fix is part of the EXTRA-FIX package. It is only enabled if LAMMPS was built with that package. See the *Build package* page for more info.

2.123.6 Related commands

dynamical_matrix, fix numdiff/virial,

2.123.7 Default

none

2.124 fix numdiff/virial command

2.124.1 Syntax

```
fix ID group-ID numdiff/virial Nevery delta
```

- ID, group-ID are documented in [fix](#) command
- numdiff/virial = style name of this fix command
- Nevery = calculate virial by finite difference every this many timesteps
- delta = magnitude of strain fields (dimensionless)

2.124.2 Examples

```
fix 1 all numdiff/stress 10 1e-6
```

2.124.3 Description

Added in version 17Feb2022.

Calculate the virial stress tensor through a finite difference calculation of energy versus strain. These values can be compared to the analytic virial tensor computed by pair styles, bond styles, etc. This can be useful for debugging or other purposes. The specified group must be “all”.

This fix applies linear strain fields of magnitude *delta* to all the atoms relative to a point at the center of the box. The strain fields are in six different directions, corresponding to the six Cartesian components of the stress tensor defined by LAMMPS. For each direction it applies the strain field in both the positive and negative senses, and the new energy of the entire system is calculated after each. The difference in these two energies divided by two times *delta*, approximates the corresponding component of the virial stress tensor, after applying a suitable unit conversion.

Note

It is important to choose a suitable value for *delta*, the magnitude of strains that are used to generate finite difference approximations to the exact virial stress. For typical systems, a value in the range of 1 part in 1e5 to 1e6 will be sufficient. However, the best value will depend on a multitude of factors including the stiffness of the interatomic potential, the thermodynamic state of the material being probed, and so on. The only way to be sure that you have made a good choice is to do a sensitivity study on a representative atomic configuration, sweeping over a wide range of values of *delta*. If *delta* is too small, the output values will vary erratically due to truncation effects. If *delta* is increased beyond a certain point, the output values will start to vary smoothly with *delta*, due to growing

contributions from higher order derivatives. In between these two limits, the numerical virial values should be largely independent of delta.

The *Nevery* argument specifies on what timesteps the force will be used calculated by finite difference.

The *delta* argument specifies the size of the displacement each atom will undergo.

2.124.4 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 which can be accessed by various *output commands*, which stores the components of the virial stress tensor as calculated by finite difference. The global vector can only be accessed on timesteps that are multiples of *Nevery* since that is when the finite difference virial is calculated. See the examples in *examples/numdiff* directory to see how this fix can be used to directly compare with the analytic virial stress tensor computed by LAMMPS.

The order of the virial stress tensor components is *xx*, *yy*, *zz*, *yz*, *xz*, and *xy*, consistent with Voigt notation. Note that the vector produced by *compute pressure* uses a different ordering, with *yz* and *xy* swapped.

The vector values calculated by this compute are “intensive”. The vector values will be in pressure *units*.

No parameter of this fix can be used with the *start/stop* keywords of the *run* command. This fix is invoked during *energy minimization*.

2.124.5 Restrictions

This fix is part of the EXTRA-FIX package. It is only enabled if LAMMPS was built with that package. See the *Build package* page for more info.

2.124.6 Related commands

fix numdiff, *compute pressure*

2.124.7 Default

none

2.125 fix nve command

Accelerator Variants: *nve/gpu*, *nve/intel*, *nve/kk*, *nve/omp*

2.125.1 Syntax

```
fix ID group-ID nve
```

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

2.125.2 Examples

```
fix 1 all nve
```

2.125.3 Description

Perform plain time integration to update position and velocity for atoms in the group each timestep. This creates a system trajectory consistent with the microcanonical ensemble (NVE) provided there are (full) periodic boundary conditions and no other “manipulations” of the system (e.g. fixes that modify forces or velocities).

This fix invokes the velocity form of the Stoermer-Verlet time integration algorithm (velocity-Verlet). Other time integration options can be invoked using the [run_style](#) command.

Styles with a *gpu*, *intel*, *kk*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed on the [Accelerator packages](#) page. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the GPU, INTEL, KOKKOS, OPENMP, and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the [Build package](#) page for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the [-suffix command-line switch](#) when you invoke LAMMPS, or you can use the [suffix](#) command in your input script.

See the [Accelerator packages](#) page for more instructions on how to use the accelerated styles effectively.

2.125.4 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](#).

2.125.5 Restrictions

none

2.125.6 Related commands

fix nvt, fix npt, run_style

2.125.7 Default

none

2.126 fix nve/asphere command

Accelerator Variants: *nve/asphere/gpu, nve/asphere/intel*

2.126.1 Syntax

```
fix ID group-ID nve/asphere
```

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

2.126.2 Examples

```
fix 1 all nve/asphere
```

2.126.3 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.

2.126.4 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](#).

Styles with a *gpu*, *intel*, *kk*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed on the [Accelerator packages](#) page. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the GPU, INTEL, KOKKOS, OPENMP, and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the [Build package](#) page for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the [-suffix command-line switch](#) when you invoke LAMMPS, or you can use the [suffix](#) command in your input script.

See the [Accelerator packages](#) page for more instructions on how to use the accelerated styles effectively.

2.126.5 Restrictions

This fix is part of the ASSPHERE package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

This fix requires that atoms store torque and angular momentum 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.

2.126.6 Related commands

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

2.126.7 Default

none

2.127 fix nve/asphere/noforce command

2.127.1 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

2.127.2 Examples

```
fix 1 all nve/asphere/noforce
```

2.127.3 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 angular momenta are used to update their positions and orientation.

This is useful as an implicit time integrator for Fast Lubrication Dynamics, since the velocity and angular momentum are updated by the [pair_style lubricateU](#) command.

2.127.4 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](#).

2.127.5 Restrictions

This fix is part of the ASPHERE package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

This fix requires that atoms store torque and angular momentum 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.

2.127.6 Related commands

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

2.127.7 Default

none

2.128 fix nve/awpmd command

2.128.1 Syntax

```
fix ID group-ID nve/awpmd
```

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

2.128.2 Examples

```
fix 1 all nve/awpmd
```

2.128.3 Description

Perform constant NVE integration to update position and velocity for nuclei and electrons in the group for the *Anantisymmetrized Wave Packet Molecular Dynamics* model. V is volume; E is energy. This creates a system trajectory consistent with the microcanonical ensemble.

The operation of this fix is exactly like that described by the [fix nve](#) command, except that the width and width-velocity of the electron wave functions are also updated.

2.128.4 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](#).

2.128.5 Restrictions

This fix is part of the AWPMD package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

2.128.6 Related commands

[fix nve](#)

2.128.7 Default

none

2.129 fix nve/body command

2.129.1 Syntax

```
fix ID group-ID nve/body
```

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

2.129.2 Examples

```
fix 1 all nve/body
```

2.129.3 Description

Perform constant NVE integration to update position, velocity, orientation, and angular velocity for body particles in the group each timestep. V is volume; E is energy. This creates a system trajectory consistent with the microcanonical ensemble. See the [Howto body](#) page for more details on using body particles.

This fix differs from the [fix nve](#) command, which assumes point particles and only updates their position and velocity.

2.129.4 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](#).

2.129.5 Restrictions

This fix is part of the BODY package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

This fix requires that atoms store torque and angular momentum and a quaternion as defined by the [atom_style body](#) command.

All particles in the group must be body particles. They cannot be point particles.

2.129.6 Related commands

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

2.129.7 Default

none

2.130 fix nve/bpm/sphere command

2.130.1 Syntax

```
fix ID group-ID nve/bpm/sphere
```

- ID, group-ID are documented in [fix](#) command
- nve/bpm/sphere = style name of this fix command
- zero or more keyword/value pairs may be appended
- keyword = *disc*

disc value = none = treat particles as 2d discs, not spheres

2.130.2 Examples

```
fix 1 all nve/bpm/sphere  
fix 1 all nve/bpm/sphere disc
```

2.130.3 Description

Added in version 4May2022.

Perform constant NVE integration to update position, velocity, angular velocity, and quaternion orientation 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. It also differs from the [fix nve/sphere](#) command which assumes finite-size spheroid particles which do not store a quaternion. It thus does not update a particle's orientation or quaternion.

If the *disc* keyword is used, then each particle is treated as a 2d disc (circle) instead of as a sphere. This is only possible for 2d simulations, as defined by the [dimension](#) keyword. The only difference between discs and spheres in this context is their moment of inertia, as used in the time integration.

2.130.4 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](#).

2.130.5 Restrictions

This fix is part of the BPM package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

This fix requires that atoms store torque, angular velocity (*omega*), a radius, and a quaternion as defined by the [atom_style bpm/sphere](#) command.

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

Use of the *disc* keyword is only allowed for 2d simulations, as defined by the [dimension](#) keyword.

2.130.6 Related commands

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

2.130.7 Default

none

2.131 fix nve/dot command

2.131.1 Syntax

```
fix ID group-ID nve/dot
```

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

2.131.2 Examples

```
fix 1 all nve/dot
```

2.131.3 Description

Apply a rigid-body integrator as described in ([Davidchack](#)) to a group of atoms, but without Langevin dynamics. This command performs Molecular dynamics (MD) via a velocity-Verlet algorithm and an evolution operator that rotates the quaternion degrees of freedom, similar to the scheme outlined in ([Miller](#)).

This command is the equivalent of the [fix nve/dotc/langevin](#) without damping and noise and can be used to determine the stability range in a NVE ensemble prior to using the Langevin-type DOTC-integrator (see also [fix nve/dotc/langevin](#)). The command is equivalent to the [fix nve](#). The particles are always considered to have a finite size.

An example input file can be found in /examples/PACKAGES/cgdna/examples/duplex1/. Further details of the implementation and stability of the integrator are contained in ([Henrich](#)). The preprint version of the article can be found [here](#).

2.131.4 Restrictions

These pair styles can only be used if LAMMPS was built with the [CG-DNA](#) package and the MOLECULE and AS-PHERE package. See the [Build package](#) page for more info.

2.131.5 Related commands

[fix nve/dotc/langevin](#), [fix nve](#)

2.131.6 Default

none

(**Davidchack**) R.L Davidchack, T.E. Ouldridge, and M.V. Tretyakov. J. Chem. Phys. 142, 144114 (2015).

(**Miller**) T. F. Miller III, M. Eleftheriou, P. Patnaik, A. Ndirango, G. J. Martyna, J. Chem. Phys., 116, 8649-8659 (2002).

(**Henrich**) O. Henrich, Y. A. Gutierrez-Fosado, T. Curk, T. E. Ouldridge, Eur. Phys. J. E 41, 57 (2018).

2.132 fix nve/dotc/langevin command

2.132.1 Syntax

```
fix ID group-ID nve/dotc/langevin Tstart Tstop damp seed keyword value
```

- ID, group-ID are documented in [fix](#) command
- nve/dotc/langevin = style name of this fix command
- Tstart,Tstop = desired temperature at start/end of run (temperature units)
- damp = damping parameter (time units)
- seed = random number seed to use for white noise (positive integer)
- keyword = *angmom*
angmom value = factor
factor = do thermostat rotational degrees of freedom via the angular momentum and apply [angmom](#)
→ numeric scale factor as discussed below

2.132.2 Examples

```
fix 1 all nve/dotc/langevin 1.0 1.0 0.03 457145 angmom 10
fix 1 all nve/dotc/langevin 0.1 0.1 78.9375 457145 angmom 10
```

2.132.3 Description

Apply a rigid-body Langevin-type integrator of the kind “Langevin C” as described in ([Davidchack](#)) to a group of atoms, which models an interaction with an implicit background solvent. This command performs Brownian dynamics (BD) via a technique that splits the integration into a deterministic Hamiltonian part and the Ornstein-Uhlenbeck process for noise and damping. The quaternion degrees of freedom are updated through an evolution operator which performs a rotation in quaternion space, preserves the quaternion norm and is akin to ([Miller](#)).

In terms of syntax this command has been closely modelled on the [fix langevin](#) and its *angmom* option. But it combines the [fix nve](#) and the [fix langevin](#) in one single command. The main feature is improved stability over the standard integrator, permitting slightly larger timestep sizes.

Note

Unlike the [fix langevin](#) this command performs also time integration of the translational and quaternion degrees of freedom.

The total force on each atom will have the form:

$$\begin{aligned} F &= F_c + F_f + F_r \\ F_f &= -\frac{m}{\text{damp}} v \\ F_r &\propto \sqrt{\frac{k_B T m}{dt \text{ damp}}} \end{aligned}$$

F_c is the conservative force computed via the usual inter-particle interactions ([pair_style](#), [bond_style](#), etc). The F_f and F_r terms are implicitly taken into account by this fix on a per-particle basis.

F_f is a frictional drag or viscous damping term proportional to the particle's velocity. The proportionality constant for each atom is computed as $\frac{m}{\text{damp}}$, where m is the mass of the particle and damp is the damping factor specified by the user.

F_r is a force due to solvent atoms at a temperature T randomly bumping into the particle. As derived from the fluctuation/dissipation theorem, its magnitude as shown above is proportional to $\sqrt{\frac{k_B T m}{dt \text{ damp}}}$, where k_B is the Boltzmann constant, T is the desired temperature, m is the mass of the particle, dt is the timestep size, and damp is the damping factor. Random numbers are used to randomize the direction and magnitude of this force as described in ([Dunweg](#)), where a uniform random number is used (instead of a Gaussian random number) for speed.

Tstart and *Tstop* have to be constant values, i.e. they cannot be variables. If used together with the oxDNA force field for coarse-grained simulation of DNA please note that $T = 0.1$ in oxDNA units corresponds to $T = 300$ K.

The *damp* parameter is specified in time units and determines how rapidly the temperature is relaxed. For example, a value of 0.03 means to relax the temperature in a timespan of (roughly) 0.03 time units τ (see the [units](#) command). The damp factor can be thought of as inversely related to the viscosity of the solvent, i.e. a small relaxation time implies a high-viscosity solvent and vice versa. See the discussion about gamma and viscosity in the documentation for the [fix viscous](#) command for more details. Note that the value 78.9375 in the second example above corresponds to a diffusion constant, which is about an order of magnitude larger than realistic ones. This has been used to sample configurations faster in Brownian dynamics simulations.

The random # *seed* must be a positive integer. A Marsaglia random number generator is used. Each processor uses the input seed to generate its own unique seed and its own stream of random numbers. Thus the dynamics of the system will not be identical on two runs on different numbers of processors.

The keyword/value option has to be used in the following way:

This fix has to be used together with the *angmom* keyword. The particles are always considered to have a finite size. The keyword *angmom* enables thermostating of the rotational degrees of freedom in addition to the usual translational degrees of freedom.

The scale factor after the *angmom* keyword gives the ratio of the rotational to the translational friction coefficient.

An example input file can be found in examples/PACKAGES/cgdna/examples/duplex2/. Further details of the implementation and stability of the integrators are contained in ([Henrich](#)). The preprint version of the article can be found [here](#).

2.132.4 Restrictions

These pair styles can only be used if LAMMPS was built with the *CG-DNA* package and the MOLECULE and AS-PHERE package. See the [Build package](#) page for more info.

2.132.5 Related commands

fix nve, *fix langevin*, *fix nve/dot*, *bond_style oxdna/fene*, *bond_style oxdna2/fene*, *pair_style oxdna/excv*, *pair_style oxdna2/excv*

2.132.6 Default

none

(Davidchack) R.L Davidchack, T.E. Ouldridge, M.V. Tretyakov. J. Chem. Phys. 142, 144114 (2015).

(Miller) T. F. Miller III, M. Eleftheriou, P. Pattnaik, A. Ndirango, G. J. Martyna, J. Chem. Phys., 116, 8649-8659 (2002).

(Dunweg) B. Dunweg, W. Paul, Int. J. Mod. Phys. C, 2, 817-27 (1991).

(Henrich) O. Henrich, Y. A. Gutierrez-Fosado, T. Curk, T. E. Ouldridge, Eur. Phys. J. E 41, 57 (2018).

2.133 fix nve/eff command

2.133.1 Syntax

```
fix ID group-ID nve/eff
```

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

2.133.2 Examples

```
fix 1 all nve/eff
```

2.133.3 Description

Perform constant NVE integration to update position and velocity for nuclei and electrons in the group for the *electron force field* model. V is volume; E is energy. This creates a system trajectory consistent with the microcanonical ensemble.

The operation of this fix is exactly like that described by the [fix nve](#) command, except that the radius and radial velocity of electrons are also updated.

2.133.4 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](#).

2.133.5 Restrictions

This fix is part of the EFF package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

2.133.6 Related commands

[fix nve](#), [fix nvt/eff](#), [fix npt/eff](#)

2.133.7 Default

none

2.134 fix nve/limit command

2.134.1 Syntax

```
fix ID group-ID nve/limit xmax
```

- ID, group-ID are documented in [fix](#) command
- nve = style name of this fix command
- xmax = maximum distance an atom can move in one timestep (distance units)

2.134.2 Examples

```
fix 1 all nve/limit 0.1
```

2.134.3 Description

Perform constant NVE updates of position and velocity for atoms in the group each timestep. A limit is imposed on the maximum distance an atom can move in one timestep. This is useful when starting a simulation with a configuration containing highly overlapped atoms. Normally this would generate huge forces which would blow atoms out of the simulation box, causing LAMMPS to stop with an error.

Using this fix can overcome that problem. Forces on atoms must still be computable (which typically means two atoms must have a separation distance > 0.0). But large velocities generated by large forces are reset to a value that corresponds to a displacement of length $xmax$ in a single timestep. $Xmax$ is specified in distance units; see the [units](#) command for details. The value of $xmax$ should be consistent with the neighbor skin distance and the frequency of neighbor list re-building, so that pairwise interactions are not missed on successive timesteps as atoms move. See the [neighbor](#) and [neigh_modify](#) commands for details.

Note that if a velocity reset occurs the integrator will not conserve energy. On steps where no velocity resets occur, this integrator is exactly like the [fix nve](#) command. Since forces are unaltered, pressures computed by thermodynamic output will still be very large for overlapped configurations.

Note

You should not use [fix shake](#) in conjunction with this fix. That is because fix shake applies constraint forces based on the predicted positions 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.

2.134.4 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](#).

2.134.5 Restrictions

none

2.134.6 Related commands

[fix nve](#), [fix nve/noforce](#), [pair_style soft](#)

2.134.7 Default

none

2.135 fix nve/line command

2.135.1 Syntax

```
fix ID group-ID nve/line
```

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

2.135.2 Examples

```
fix 1 all nve/line
```

2.135.3 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 [Howto spherical](#) page 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.

2.135.4 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](#).

2.135.5 Restrictions

This fix is part of the ASSPHERE package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

This fix requires that particles be line segments as defined by the [atom_style line](#) command.

2.135.6 Related commands

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

2.135.7 Default

none

2.136 fix nve/manifold/rattle command

2.136.1 Syntax

```
fix ID group-ID nve/manifold/rattle tol maxit manifold manifold-args keyword value ...
```

- ID, group-ID are documented in [fix](#) command
- nve/manifold/rattle = style name of this fix command
- tol = tolerance to which Newton iteration must converge
- maxit = maximum number of iterations to perform
- manifold = name of the manifold
- manifold-args = parameters for the manifold

- one or more keyword/value pairs may be appended

keyword = every

every values = N

N = print info about iteration every N steps. N = 0 means no output

2.136.2 Examples

```
fix 1 all nve/manifold/rattle 1e-4 10 sphere 5.0
fix step all nve/manifold/rattle 1e-8 100 ellipsoid 2.5 2.5 5.0 every 25
```

2.136.3 Description

Perform constant NVE integration to update position and velocity for atoms constrained to a curved surface (manifold) in the group each timestep. The constraint is handled by RATTLE ([Andersen](#)) written out for the special case of single-particle constraints as explained in ([Paquay](#)). V is volume; E is energy. This way, the dynamics of particles constrained to curved surfaces can be studied. If combined with [fix langevin](#), this generates Brownian motion of particles constrained to a curved surface. For a list of currently supported manifolds and their parameters, see the [Howto manifold](#) doc page.

Note that the particles must initially be close to the manifold in question. If not, RATTLE will not be able to iterate until the constraint is satisfied, and an error is generated. For simple manifolds this can be achieved with [region](#) and [create_atoms](#) commands, but for more complex surfaces it might be more useful to write a script.

The manifold args may be equal-style variables, like so:

```
variable R equal "ramp(5.0,3.0)"
fix shrink_sphere all nve/manifold/rattle 1e-4 10 sphere v_R
```

In this case, the manifold parameter will change in time according to the variable. This is not a problem for the time integrator as long as the change of the manifold is slow with respect to the dynamics of the particles. Note that if the manifold has to exert work on the particles because of these changes, the total energy might not be conserved.

2.136.4 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](#).

2.136.5 Restrictions

This fix is part of the MANIFOLD package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

2.136.6 Related commands

`fix nvt/manifold/rattle, fix manifoldforce`

2.136.7 Default

`every = 0, tchain = 3`

(**Andersen**) Andersen, J. Comp. Phys. 52, 24, (1983).

(**Paquay**) Paquay and Kusters, Biophys. J., 110, 6, (2016). preprint available at arXiv:1411.3019.

2.137 fix nve/noforce command

2.137.1 Syntax

`fix ID group-ID nve`

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

2.137.2 Examples

`fix 3 wall nve/noforce`

2.137.3 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.

2.137.4 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](#).

2.137.5 Restrictions

none

2.137.6 Related commands

fix nve

2.137.7 Default

none

2.138 fix nve/sphere command

Accelerator Variants: *nve/sphere/omp*, *nve/sphere/kk*

2.138.1 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* or *disc*
 - update value = dipole or dipole/dlm
 - dipole = update orientation of dipole moment during integration
 - dipole/dlm = use DLM integrator to update dipole orientation
 - disc value = none = treat particles as 2d discs, not spheres

2.138.2 Examples

```
fix 1 all nve/sphere
fix 1 all nve/sphere update dipole
fix 1 all nve/sphere disc
fix 1 all nve/sphere update dipole/dlm
```

2.138.3 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 finite-size particles, e.g. spheroids via use of the [atom_style hybrid sphere dipole](#) command.

The default dipole orientation integrator can be changed to the Dullweber-Leimkuhler-McLachlan integration scheme ([Dullweber](#)) when using *update* with the value *dipole/dlm*. This integrator is symplectic and time-reversible, giving better energy conservation and allows slightly longer timesteps at only a small additional computational cost.

If the *disc* keyword is used, then each particle is treated as a 2d disc (circle) instead of as a sphere. This is only possible for 2d simulations, as defined by the [dimension](#) keyword. The only difference between discs and spheres in this context is their moment of inertia, as used in the time integration.

Styles with a *gpu*, *intel*, *kk*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed on the [Accelerator packages](#) page. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the GPU, INTEL, KOKKOS, OPENMP, and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the [Build package](#) page for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the [-suffix command-line switch](#) when you invoke LAMMPS, or you can use the [suffix](#) command in your input script.

See the [Accelerator packages](#) page for more instructions on how to use the accelerated styles effectively.

2.138.4 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](#).

2.138.5 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.

Use of the *disc* keyword is only allowed for 2d simulations, as defined by the [dimension](#) keyword.

2.138.6 Related commands

fix nve, *fix nve/asphere*

2.138.7 Default

none

(Dullweber) Dullweber, Leimkuhler and McLachlan, J Chem Phys, 107, 5840 (1997).

2.139 fix nve/spin command

2.139.1 Syntax

```
fix ID group-ID nve/spin keyword values
```

- ID, group-ID are documented in *fix* command
- nve/spin = style name of this fix command
- keyword = *lattice*

lattice value = moving or frozen

moving = integrate both spin and atomic degrees of freedom

frozen = integrate spins on a fixed lattice

2.139.2 Examples

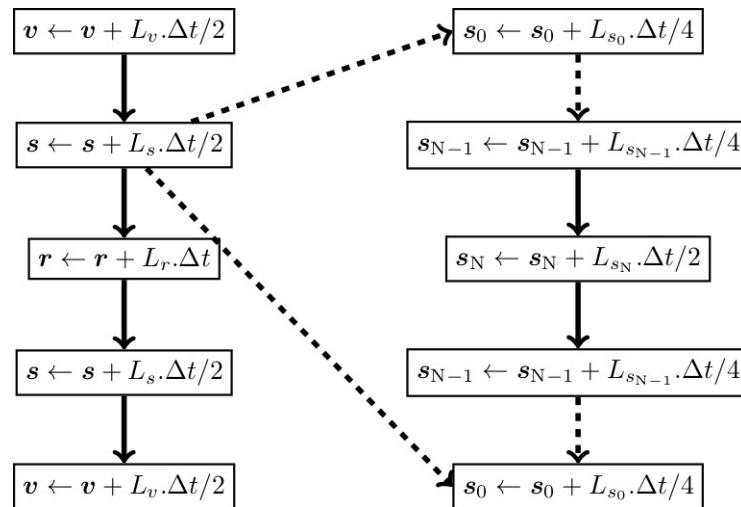
```
fix 3 all nve/spin lattice moving
fix 1 all nve/spin lattice frozen
```

2.139.3 Description

Perform a symplectic integration for the spin or spin-lattice system.

The *lattice* keyword defines if the spins are integrated on a lattice of fixed atoms (*lattice* = frozen), or if atoms are moving (*lattice* = moving). The first case corresponds to a spin dynamics calculation, and the second to a spin-lattice calculation. By default a spin-lattice integration is performed (*lattice* = moving).

The *nve/spin* fix applies a Suzuki-Trotter decomposition to the equations of motion of the spin lattice system, following the scheme:



according to the implementation reported in ([Omelyan](#)).

A sectoring method enables this scheme for parallel calculations. The implementation of this sectoring algorithm is reported in ([Tranchida](#)).

2.139.4 Restrictions

This fix style can only be used if LAMMPS was built with the SPIN package. See the [Build package](#) page for more info.

To use the spin algorithm, it is necessary to define a map with the atom_modify command. Typically, by adding the command:

```
atom_modify map array
```

before you create the simulation box. Note that the keyword “hash” instead of “array” is also valid.

2.139.5 Related commands

atom_style spin, fix nve

2.139.6 Default

The option default is lattice = moving.

(**Omelyan**) Omelyan, Mryglod, and Folk. Phys. Rev. Lett. 86(5), 898. (2001).

(**Tranchida**) Tranchida, Plimpton, Thibaudeau and Thompson, Journal of Computational Physics, 372, 406-425, (2018).

2.140 fix nve/tri command

2.140.1 Syntax

```
fix ID group-ID nve/tri
```

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

2.140.2 Examples

```
fix 1 all nve/tri
```

2.140.3 Description

Perform constant NVE integration to update position, velocity, orientation, and angular momentum for triangular particles in the group each timestep. V is volume; E is energy. This creates a system trajectory consistent with the microcanonical ensemble. See the [Howto spherical](#) page for an overview of using triangular particles.

This fix differs from the [fix nve](#) command, which assumes point particles and only updates their position and velocity.

2.140.4 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](#).

2.140.5 Restrictions

This fix is part of the ASPHERE package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

This fix requires that particles be triangles as defined by the [atom_style tri](#) command.

2.140.6 Related commands

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

2.140.7 Default

none

2.141 fix nvk command

2.141.1 Syntax

```
fix ID group-ID nvk
```

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

2.141.2 Examples

```
fix 1 all nvk
```

2.141.3 Description

Perform constant kinetic energy integration using the Gaussian thermostat to update position and velocity for atoms in the group each timestep. V is volume; K is kinetic energy. This creates a system trajectory consistent with the isokinetic ensemble.

The equations of motion used are those of Minary et al in ([Minary](#)), a variant of those initially given by Zhang in ([Zhang](#)).

The kinetic energy will be held constant at its value given when fix nvk is initiated. If a different kinetic energy is desired, the [velocity](#) command should be used to change the kinetic energy prior to this fix.

2.141.4 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](#).

2.141.5 Restrictions

The Gaussian thermostat only works when it is applied to all atoms in the simulation box. Therefore, the group must be set to all.

This fix has not yet been implemented to work with the RESPA integrator.

This fix is part of the EXTRA-FIX package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

2.141.6 Related commands

none

2.141.7 Default

none

(**Minary**) Minary, Martyna, and Tuckerman, J Chem Phys, 18, 2510 (2003).

(**Zhang**) Zhang, J Chem Phys, 106, 6102 (1997).

2.142 fix nvt/asphere command

Accelerator Variants: *nvt/asphere/omp*

2.142.1 Syntax

```
fix ID group-ID nvt/asphere keyword value ...
```

- ID, group-ID are documented in [fix](#) command
- nvt/asphere = style name of this fix command
- additional thermostat related keyword/value pairs from the [fix nvt](#) command can be appended

2.142.2 Examples

```
fix 1 all nvt/asphere temp 300.0 300.0 100.0
fix 1 all nvt/asphere temp 300.0 300.0 100.0 drag 0.2
```

2.142.3 Description

Perform constant NVT integration to update position, velocity, orientation, and angular velocity each timestep for aspherical or ellipsoidal particles in the group using a Nose/Hoover temperature thermostat. V is volume; T is temperature. This creates a system trajectory consistent with the canonical ensemble.

This fix differs from the [fix nvt](#) command, which assumes point particles and only updates their position and velocity.

The thermostat is applied to both the translational and rotational degrees of freedom for the aspherical particles, assuming a compute is used which calculates a temperature that includes the rotational degrees of freedom (see below). The translational degrees of freedom can also have a bias velocity removed from them before thermostating takes place; see the description below.

Additional parameters affecting the thermostat are specified by keywords and values documented with the [fix nvt](#) command. See, for example, discussion of the *temp* and *drag* keywords.

This fix computes a temperature each timestep. To do this, the fix creates its own compute of style “temp/asphere”, as if this command had been issued:

```
compute fix-ID_temp group-ID temp/asphere
```

See the [compute temp/asphere](#) command for details. Note that the ID of the new compute is the fix-ID + underscore + “temp”, and the group for the new compute is the same as the fix group.

Note that this is NOT the compute used by thermodynamic output (see the [thermo_style](#) command) with ID = *thermo_temp*. This means you can change the attributes of this fix’s temperature (e.g. its degrees-of-freedom) via the [compute_modify](#) command or print this temperature during thermodynamic output via the [thermo_style custom](#) command using the appropriate compute-ID. It also means that changing attributes of *thermo_temp* will have no effect on this fix.

Like other fixes that perform thermostating, this fix can be used with [compute commands](#) that remove a “bias” from the atom velocities. E.g. to apply the thermostat only to atoms within a spatial [region](#), or to remove the center-of-mass velocity from a group of atoms, or to remove the x-component of velocity from the calculation.

This is not done by default, but only if the [fix_modify](#) command is used to assign a temperature compute to this fix that includes such a bias term. See the doc pages for individual [compute temp commands](#) to determine which ones include a bias. In this case, the thermostat works in the following manner: bias is removed from each atom, thermostating is performed on the remaining thermal degrees of freedom, and the bias is added back in.

Styles with a *gpu*, *intel*, *kk*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed on the [Accelerator packages](#) page. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the GPU, INTEL, KOKKOS, OPENMP, and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the [Build package](#) page for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the [-suffix command-line switch](#) when you invoke LAMMPS, or you can use the [suffix](#) command in your input script.

See the [Accelerator packages](#) page for more instructions on how to use the accelerated styles effectively.

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

This fix writes the state of the Nose/Hoover thermostat to [binary restart files](#). See the [read_restart](#) command for info on how to re-specify a fix in an input script that reads a restart file, so that the operation of the fix continues in an uninterrupted fashion.

The [fix_modify temp](#) option is supported by this fix. You can use it to assign a [compute](#) you have defined to this fix which will be used in its thermostating procedure.

The cumulative energy change in the system imposed by this fix is included in the [thermodynamic output](#) keywords *ecouple* and *econserve*. See the [thermo_style](#) doc page for details.

This fix computes the same global scalar and global vector of quantities as does the [fix nvt](#) command.

This fix can ramp its target temperature over multiple runs, using the *start* and *stop* keywords of the [run](#) command. See the [run](#) command for details of how to do this.

This fix is not invoked during [energy minimization](#).

2.142.5 Restrictions

This fix is part of the ASSPHERE package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

This fix requires that atoms store torque and angular momentum 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.

2.142.6 Related commands

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

2.142.7 Default

none

2.143 fix nvt/body command

2.143.1 Syntax

```
fix ID group-ID nvt/body keyword value ...
```

- ID, group-ID are documented in [fix](#) command
- nvt/body = style name of this fix command
- additional thermostat related keyword/value pairs from the [fix nvt](#) command can be appended

2.143.2 Examples

```
fix 1 all nvt/body temp 300.0 300.0 100.0  
fix 1 all nvt/body temp 300.0 300.0 100.0 drag 0.2
```

2.143.3 Description

Perform constant NVT integration to update position, velocity, orientation, and angular velocity each timestep for body particles in the group using a Nose/Hoover temperature thermostat. V is volume; T is temperature. This creates a system trajectory consistent with the canonical ensemble.

This fix differs from the [fix nvt](#) command, which assumes point particles and only updates their position and velocity.

The thermostat is applied to both the translational and rotational degrees of freedom for the body particles, assuming a compute is used which calculates a temperature that includes the rotational degrees of freedom (see below). The translational degrees of freedom can also have a bias velocity removed from them before thermostating takes place; see the description below.

Additional parameters affecting the thermostat are specified by keywords and values documented with the [fix nvt](#) command. See, for example, discussion of the *temp* and *drag* keywords.

This fix computes a temperature each timestep. To do this, the fix creates its own compute of style “temp/body”, as if this command had been issued:

```
compute fix-ID_temp group-ID temp/body
```

See the [compute temp/body](#) command for details. Note that the ID of the new compute is the fix-ID + underscore + “temp”, and the group for the new compute is the same as the fix group.

Note that this is NOT the compute used by thermodynamic output (see the [thermo_style](#) command) with ID = *thermo_temp*. This means you can change the attributes of this fix’s temperature (e.g. its degrees-of-freedom) via the [compute_modify](#) command or print this temperature during thermodynamic output via the [thermo_style custom](#) command using the appropriate compute-ID. It also means that changing attributes of *thermo_temp* will have no effect on this fix.

Like other fixes that perform thermostating, this fix can be used with [compute commands](#) that remove a “bias” from the atom velocities. E.g. to apply the thermostat only to atoms within a spatial [region](#), or to remove the center-of-mass velocity from a group of atoms, or to remove the x-component of velocity from the calculation.

This is not done by default, but only if the [fix_modify](#) command is used to assign a temperature compute to this fix that includes such a bias term. See the doc pages for individual [compute temp commands](#) to determine which ones include a bias. In this case, the thermostat works in the following manner: bias is removed from each atom, thermostating is performed on the remaining thermal degrees of freedom, and the bias is added back in.

Styles with a *gpu*, *intel*, *kk*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed on the [Accelerator packages](#) page. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the GPU, INTEL, KOKKOS, OPENMP, and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the [Build package](#) page for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the [-suffix command-line switch](#) when you invoke LAMMPS, or you can use the [suffix](#) command in your input script.

See the [Accelerator packages](#) page for more instructions on how to use the accelerated styles effectively.

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

This fix writes the state of the Nose/Hoover thermostat to [binary restart files](#). See the [read_restart](#) command for info on how to re-specify a fix in an input script that reads a restart file, so that the operation of the fix continues in an uninterrupted fashion.

The [fix_modify temp](#) option is supported by this fix. You can use it to assign a [compute](#) you have defined to this fix which will be used in its thermostating procedure.

The cumulative energy change in the system imposed by this fix is included in the [thermodynamic output](#) keywords *ecouple* and *econserve*. See the [thermo_style](#) doc page for details.

This fix computes the same global scalar and global vector of quantities as does the [fix nvt](#) command.

This fix can ramp its target temperature over multiple runs, using the *start* and *stop* keywords of the [run](#) command. See the [run](#) command for details of how to do this.

This fix is not invoked during [energy minimization](#).

2.143.5 Restrictions

This fix is part of the BODY package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

This fix requires that atoms store torque and angular momentum and a quaternion as defined by the [atom_style body](#) command.

2.143.6 Related commands

[fix nvt](#), [fix nve_body](#), [fix npt_body](#), [fix_modify](#)

2.143.7 Default

none

2.144 fix nvt/manifold/rattle command

2.144.1 Syntax

```
fix ID group-ID nvt/manifold/rattle tol maxit manifold manifold-args keyword value ...
```

- ID, group-ID are documented in [fix](#) command
- nvt/manifold/rattle = style name of this fix command
- tol = tolerance to which Newton iteration must converge
- maxit = maximum number of iterations to perform
- manifold = name of the manifold
- manifold-args = parameters for the manifold
- one or more keyword/value pairs may be appended

keyword = temp or tchain or every

temp values = Tstart Tstop Tdamp

Tstart, Tstop = external temperature at start/end of run

Tdamp = temperature damping parameter (time units)

tchain value = N

N = length of thermostat chain (1 = single thermostat)

every value = N

N = print info about iteration every N steps. N = 0 means no output

2.144.2 Examples

```
fix 1 all nvt/manifold/rattle 1e-4 10 cylinder 3.0 temp 1.0 1.0 10.0
```

2.144.3 Description

This fix combines the RATTLE-based ([Andersen](#)) time integrator of [fix nve/manifold/rattle \(Paquay\)](#) with a Nose-Hoover-chain thermostat to sample the canonical ensemble of particles constrained to a curved surface (manifold). This sampling does suffer from discretization bias of $O(dt)$. For a list of currently supported manifolds and their parameters, see the [Howto manifold](#) doc page.

2.144.4 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](#).

2.144.5 Restrictions

This fix is part of the MANIFOLD package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

2.144.6 Related commands

[fix nve/manifold/rattle](#), [fix manifoldforce](#) **Default:** every = 0

([Andersen](#)) Andersen, J. Comp. Phys. 52, 24, (1983).

([Paquay](#)) Paquay and Kusters, Biophys. J., 110, 6, (2016). preprint available at arXiv:1411.3019.

2.145 fix nvt/sllod command

Accelerator Variants: [nvt/sllod/intel](#), [nvt/sllod/omp](#), [nvt/sllod/kk](#)

2.145.1 Syntax

```
fix ID group-ID nvt/slloid keyword value ...
```

- ID, group-ID are documented in [fix](#) command
 - nvt/slloid = style name of this fix command
 - zero or more keyword/value pairs may be appended
- keyword = psllod
 psllod value = no or yes = use SLLOD or p-SLLOD variant, respectively
- additional thermostat related keyword/value pairs from the [fix nvt](#) command can be appended

2.145.2 Examples

```
fix 1 all nvt/slloid temp 300.0 300.0 100.0
fix 1 all nvt/slloid temp 300.0 300.0 100.0 drag 0.2
```

2.145.3 Description

Perform constant NVT integration to update positions and velocities each timestep for atoms in the group using a Nose/Hoover temperature thermostat. V is volume; T is temperature. This creates a system trajectory consistent with the canonical ensemble.

This thermostat is used for a simulation box that is changing size and/or shape, for example in a non-equilibrium MD (NEMD) simulation. The size/shape change is induced by use of the [fix deform](#) command, so each point in the simulation box can be thought of as having a “streaming” velocity. This position-dependent streaming velocity is subtracted from each atom’s actual velocity to yield a thermal velocity which is used for temperature computation and thermostating. For example, if the box is being sheared in x, relative to y, then points at the bottom of the box (low y) have a small x velocity, while points at the top of the box (hi y) have a large x velocity. These velocities do not contribute to the thermal “temperature” of the atom.

Note

[Fix deform](#) has an option for remapping either atom coordinates or velocities to the changing simulation box. To use fix nvt/slloid, fix deform should NOT remap atom positions, because fix nvt/slloid adjusts the atom positions and velocities to create a velocity profile that matches the changing box size/shape. Fix deform SHOULD remap atom velocities when atoms cross periodic boundaries since that is consistent with maintaining the velocity profile created by fix nvt/slloid. LAMMPS will give an error if this setting is not consistent.

The SLLOD equations of motion, originally proposed by Hoover and Ladd (see ([Evans and Morriss](#))), were proven to be equivalent to Newton’s equations of motion for shear flow by ([Evans and Morriss](#)). They were later shown to generate the desired velocity gradient and the correct production of work by stresses for all forms of homogeneous flow by ([Daivis and Todd](#)).

Changed in version 8Feb2023.

For the default (*psllod = no*), the LAMMPS implementation adheres to the standard SLLOD equations of motion, as defined by ([Evans and Morriss](#)). The option *psllod = yes* invokes the slightly different SLLOD variant first introduced by ([Tuckerman et al.](#)) as g-SLLOD and later by ([Edwards](#)) as p-SLLOD. In all cases, the equations of motion are coupled to a Nose/Hoover chain thermostat in a velocity Verlet formulation, closely following the implementation used for the [fix nvt](#) command.

Note

A recent (2017) book by (*Todd and Daivis*) discusses use of the SLLOD method and non-equilibrium MD (NEMD) thermostatting generally, for both simple and complex fluids, e.g. molecular systems. The latter can be tricky to do correctly.

Additional parameters affecting the thermostat are specified by keywords and values documented with the [fix nvt](#) command. See, for example, discussion of the *temp* and *drag* keywords.

This fix computes a temperature each timestep. To do this, the fix creates its own compute of style “temp/deform”, as if this command had been issued:

```
compute fix-ID temp group-ID temp/deform
```

See the [compute temp/deform](#) command for details. Note that the ID of the new compute is the fix-ID + underscore + “temp”, and the group for the new compute is the same as the fix group.

Note that this is NOT the compute used by thermodynamic output (see the [thermo_style](#) command) with ID = *thermo_temp*. This means you can change the attributes of this fix’s temperature (e.g. its degrees-of-freedom) via the [compute_modify](#) command or print this temperature during thermodynamic output via the [thermo_style custom](#) command using the appropriate compute-ID. It also means that changing attributes of *thermo_temp* will have no effect on this fix.

Like other fixes that perform thermostatting, this fix can be used with [compute commands](#) that remove a “bias” from the atom velocities. E.g. to apply the thermostat only to atoms within a spatial [region](#), or to remove the center-of-mass velocity from a group of atoms, or to remove the x-component of velocity from the calculation.

This is not done by default, but only if the [fix_modify](#) command is used to assign a temperature compute to this fix that includes such a bias term. See the doc pages for individual [compute temp commands](#) to determine which ones include a bias. In this case, the thermostat works in the following manner: bias is removed from each atom, thermostatting is performed on the remaining thermal degrees of freedom, and the bias is added back in.

Styles with a *gpu*, *intel*, *kk*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed on the [Accelerator packages](#) page. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the GPU, INTEL, KOKKOS, OPENMP, and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the [Build package](#) page for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the [-suffix command-line switch](#) when you invoke LAMMPS, or you can use the [suffix](#) command in your input script.

See the [Accelerator packages](#) page for more instructions on how to use the accelerated styles effectively.

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

This fix writes the state of the Nose/Hoover thermostat to *binary restart files*. See the [read_restart](#) command for info on how to re-specify a fix in an input script that reads a restart file, so that the operation of the fix continues in an uninterrupted fashion.

The [fix_modify temp](#) option is supported by this fix. You can use it to assign a [compute](#) you have defined to this fix which will be used in its thermostatting procedure.

The cumulative energy change in the system imposed by this fix is included in the [thermodynamic output](#) keywords *ecouple* and *econserve*. See the [thermo_style](#) doc page for details.

This fix computes the same global scalar and global vector of quantities as does the [fix nvt](#) command.

This fix can ramp its target temperature over multiple runs, using the *start* and *stop* keywords of the [run](#) command. See the [run](#) command for details of how to do this.

This fix is not invoked during *energy minimization*.

2.145.5 Restrictions

This fix works best without Nose-Hoover chain thermostats, i.e. using *tchain* = 1. Setting *tchain* to larger values can result in poor equilibration.

2.145.6 Related commands

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

2.145.7 Default

Same as [fix nvt](#), except *tchain* = 1, *psllod* = *no*.

(Evans and Morriss) Evans and Morriss, Phys Rev A, 30, 1528 (1984).

(Daivis and Todd) Daivis and Todd, J Chem Phys, 124, 194103 (2006).

(Todd and Daivis) Todd and Daivis, Nonequilibrium Molecular Dynamics (book), Cambridge University Press, (2017) <https://doi.org/10.1017/9781139017848>.

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

(Edwards) Edwards, Baig, and Keffer, J Chem Phys 124, 194104 (2006).

2.146 fix nvt/sllo/d/eff command

2.146.1 Syntax

```
fix ID group-ID nvt/sllo/d/eff keyword value ...
```

- ID, group-ID are documented in [fix](#) command
- nvt/sllo/d/eff = style name of this fix command
- zero or more keyword/value pairs may be appended
 - keyword = psllod
 - psllod value = no or yes = use SLLOD or p-SLLOD variant, respectively
 - additional thermostat related keyword/value pairs from the [fix nvt/eff](#) command may be appended, too.

2.146.2 Examples

```
fix 1 all nvt/sllod/eff temp 300.0 300.0 0.1
fix 1 all nvt/sllod/eff temp 300.0 300.0 0.1 drag 0.2
```

2.146.3 Description

Perform constant NVT integration to update positions and velocities each timestep for nuclei and electrons in the group for the [electron force field](#) model, using a Nose/Hoover temperature thermostat. V is volume; T is temperature. This creates a system trajectory consistent with the canonical ensemble.

The operation of this fix is exactly like that described by the [fix nvt/sllod](#) command, except that the radius and radial velocity of electrons are also updated and thermostatted. Likewise the temperature calculated by the fix, using the compute it creates (as discussed in the [fix nvt, npt, and nph](#) doc page), is performed with a [compute temp/deform/eff](#) command that includes the eFF contribution to the temperature from the electron radial velocity.

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

This fix writes the state of the Nose/Hoover thermostat to [binary restart files](#). See the [read_restart](#) command for info on how to re-specify a fix in an input script that reads a restart file, so that the operation of the fix continues in an uninterrupted fashion.

The [fix_modify temp](#) option is supported by this fix. You can use it to assign a [compute](#) you have defined to this fix which will be used in its thermostating procedure.

The cumulative energy change in the system imposed by this fix is included in the [thermodynamic output](#) keywords [ecouple](#) and [econserve](#). See the [thermo_style](#) doc page for details.

This fix computes the same global scalar and global vector of quantities as does the [fix nvt/eff](#) command.

This fix can ramp its target temperature over multiple runs, using the [start](#) and [stop](#) keywords of the [run](#) command. See the [run](#) command for details of how to do this.

This fix is not invoked during [energy minimization](#).

2.146.5 Restrictions

This fix is part of the EFF package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

This fix works best without Nose-Hoover chain thermostats, i.e. using tchain = 1. Setting tchain to larger values can result in poor equilibration.

2.146.6 Related commands

[fix nve/eff](#), [fix nvt/eff](#), [fix langevin/eff](#), [fix nvt/sllod](#), [fix_modify](#), [compute temp/deform/eff](#)

2.146.7 Default

Same as [fix nvt/eff](#), except tchain = 1.

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

2.147 fix nvt/sphere command

Accelerator Variants: *nvt/sphere/omp*

2.147.1 Syntax

```
fix ID group-ID nvt/sphere keyword value ...
```

- ID, group-ID are documented in [fix](#) command
- nvt/sphere = style name of this fix command
- zero or more keyword/value pairs may be appended
- keyword = *disc*
disc value = none = treat particles as 2d discs, not spheres
- NOTE: additional thermostat and dipole related keyword/value pairs from the [fix nvt](#) command can be appended

2.147.2 Examples

```
fix 1 all nvt/sphere temp 300.0 300.0 100.0
fix 1 all nvt/sphere temp 300.0 300.0 100.0 disc
fix 1 all nvt/sphere temp 300.0 300.0 100.0 drag 0.2
fix 1 all nvt/sphere temp 300.0 300.0 100.0 update dipole
```

2.147.3 Description

Perform constant NVT integration to update position, velocity, and angular velocity each timestep for finite-size spherical particles in the group using a Nose/Hoover temperature thermostat. V is volume; T is temperature. This creates a system trajectory consistent with the canonical ensemble.

This fix differs from the [fix nvt](#) command, which assumes point particles and only updates their position and velocity.

The thermostat is applied to both the translational and rotational degrees of freedom for the spherical particles, assuming a compute is used which calculates a temperature that includes the rotational degrees of freedom (see below). The translational degrees of freedom can also have a bias velocity removed from them before thermostating takes place; see the description below.

If the *disc* keyword is used, then each particle is treated as a 2d disc (circle) instead of as a sphere. This is only possible for 2d simulations, as defined by the [dimension](#) keyword. The only difference between discs and spheres in this context is their moment of inertia, as used in the time integration.

Additional parameters affecting the thermostat are specified by keywords and values documented with the [fix nvt](#) command. See, for example, discussion of the *temp* and *drag* keywords.

This fix computes a temperature each timestep. To do this, the fix creates its own compute of style “temp/sphere”, as if this command had been issued:

```
compute fix-ID_temp group-ID temp/sphere
```

See the [compute temp/sphere](#) command for details. Note that the ID of the new compute is the fix-ID + underscore + “temp”, and the group for the new compute is the same as the fix group.

Note that this is NOT the compute used by thermodynamic output (see the [thermo_style](#) command) with ID = *thermo_temp*. This means you can change the attributes of this fix’s temperature (e.g. its degrees-of-freedom) via the [compute_modify](#) command or print this temperature during thermodynamic output via the [thermo_style custom](#) command using the appropriate compute-ID. It also means that changing attributes of *thermo_temp* will have no effect on this fix.

Like other fixes that perform thermostating, this fix can be used with [compute commands](#) that remove a “bias” from the atom velocities. E.g. to apply the thermostat only to atoms within a spatial [region](#), or to remove the center-of-mass velocity from a group of atoms, or to remove the x-component of velocity from the calculation.

This is not done by default, but only if the [fix_modify](#) command is used to assign a temperature compute to this fix that includes such a bias term. See the doc pages for individual [compute temp commands](#) to determine which ones include a bias. In this case, the thermostat works in the following manner: bias is removed from each atom, thermostating is performed on the remaining thermal degrees of freedom, and the bias is added back in.

Styles with a *gpu*, *intel*, *kk*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed on the [Accelerator packages](#) page. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the GPU, INTEL, KOKKOS, OPENMP, and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the [Build package](#) page for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the [-suffix command-line switch](#) when you invoke LAMMPS, or you can use the [suffix](#) command in your input script.

See the [Accelerator packages](#) page for more instructions on how to use the accelerated styles effectively.

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

This fix writes the state of the Nose/Hoover thermostat to [binary restart files](#). See the [read_restart](#) command for info on how to re-specify a fix in an input script that reads a restart file, so that the operation of the fix continues in an uninterrupted fashion.

The [fix_modify temp](#) option is supported by this fix. You can use it to assign a [compute](#) you have defined to this fix which will be used in its thermostating procedure.

The cumulative energy change in the system imposed by this fix is included in the [thermodynamic output](#) keywords *ecouple* and *econserve*. See the [thermo_style](#) doc page for details.

This fix computes the same global scalar and global vector of quantities as does the [fix nvt](#) command.

This fix can ramp its target temperature over multiple runs, using the *start* and *stop* keywords of the [run](#) command. See the [run](#) command for details of how to do this.

This fix is not invoked during [energy minimization](#).

2.147.5 Restrictions

This fix requires that atoms store torque and angular velocity (omega) and a radius as defined by the *atom_style sphere* command.

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

Use of the *disc* keyword is only allowed for 2d simulations, as defined by the *dimension* keyword.

2.147.6 Related commands

```
fix nvt,fix nve_sphere,  
fix nvt_asphere,fix npt_sphere,fix_modify
```

2.147.7 Default

none

2.148 fix oneway command

2.148.1 Syntax

```
fix ID group-ID oneway N region-ID direction
```

- ID, group-ID are documented in *fix* command
- oneway = style name of this fix command
- N = apply this fix every this many timesteps
- region-ID = ID of region where fix is active
- direction = *x* or *-x* or *y* or *-y* or *z* or *-z* = coordinate and direction of the oneway constraint

2.148.2 Examples

```
fix 1 ions oneway 10 semi -x  
fix 2 all oneway 1 left -z  
fix 3 all oneway 1 right z
```

2.148.3 Description

Enforce that particles in the group and in a given region can only move in one direction. This is done by reversing a particle's velocity component, if it has the wrong sign in the specified dimension. The effect is that the particle moves in one direction only.

This can be used, for example, as a simple model of a semi-permeable membrane, or as an implementation of Maxwell's demon.

2.148.4 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](#).

2.148.5 Restrictions

This fix is part of the EXTRA-FIX package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

2.148.6 Related commands

[fix wall/reflect](#) command

2.148.7 Default

none

2.149 fix orient/fcc command

2.150 fix orient/bcc command

2.150.1 Syntax

```
fix ID group-ID orient/fcc nstats dir alat dE cutlo cuthi file0 file1
fix ID group-ID orient/bcc nstats dir alat dE cutlo cuthi file0 file1
```

- ID, group-ID are documented in [fix](#) command
- nstats = print stats every this many steps, 0 = never
- dir = 0/1 for which crystal is used as reference
- alat = fcc/bcc cubic lattice constant (distance units)
- dE = energy added to each atom (energy units)
- cutlo,cuthi = values between 0.0 and 1.0, cutlo < cuthi
- file0,file1 = files that specify orientation of each grain

2.150.2 Examples

```
fix gb all orient/fcc 0 1 4.032008 0.001 0.25 0.75 xi.vec chi.vec
fix gb all orient/bcc 0 1 2.882 0.001 0.25 0.75 ngb.left ngb.right
```

2.150.3 Description

The fix applies an orientation-dependent force to atoms near a planar grain boundary which can be used to induce grain boundary migration (in the direction perpendicular to the grain boundary plane). The motivation and explanation of this force and its application are described in ([Janssens](#)). The adaptation to bcc crystals is described in ([Wicaksono1](#)). The computed force is only applied to atoms in the fix group.

The basic idea is that atoms in one grain (on one side of the boundary) have a potential energy dE added to them. Atoms in the other grain have 0.0 potential energy added. Atoms near the boundary (whose neighbor environment is intermediate between the two grain orientations) have an energy between 0.0 and dE added. This creates an effective driving force to reduce the potential energy of atoms near the boundary by pushing them towards one of the grain orientations. For $dir = 1$ and $dE > 0$, the boundary will thus move so that the grain described by file0 grows and the grain described by file1 shrinks. Thus this fix is designed for simulations of two-grain systems, either with one grain boundary and free surfaces parallel to the boundary, or a system with periodic boundary conditions and two equal and opposite grain boundaries. In either case, the entire system can displace during the simulation, and such motion should be accounted for in measuring the grain boundary velocity.

The potential energy added to atom I is given by these formulas

$$\xi_i = \sum_{j=1}^{12} |\mathbf{r}_j - \mathbf{r}_j^I| \quad (1)$$

$$\xi_{IJ} = \sum_{j=1}^{12} |\mathbf{r}_j^J - \mathbf{r}_j^I| \quad (2)$$

$$\xi_{low} = cutlo \xi_{IJ} \quad (3)$$

$$\xi_{high} = cuthi \xi_{IJ} \quad (4)$$

$$\omega_i = \frac{\pi}{2} \frac{\xi_i - \xi_{low}}{\xi_{high} - \xi_{low}} \quad (5)$$

$$\begin{aligned} u_i &= 0 && \text{for } \xi_i < \xi_{low} \\ &= dE \frac{1 - \cos(2\omega_i)}{2} && \text{for } \xi_{low} < \xi_i < \xi_{high} \quad (6) \\ &= dE && \text{for } \xi_{high} < \xi_i \end{aligned}$$

which are fully explained in ([Janssens](#)). For fcc crystals this order parameter Ξ_i for atom I in equation (1) is a sum over the 12 nearest neighbors of atom I. For bcc crystals it is the corresponding sum of the 8 nearest neighbors. \mathbf{r}_j is the vector from atom I to its neighbor J, and \mathbf{r}_{IJ} is a vector in the reference (perfect) crystal. That is, if $dir = 0/1$, then \mathbf{r}_{IJ} is a vector to an atom coord from file 0/1. Equation (2) gives the expected value of the order parameter Ξ_{IJ} in the other grain. Hi and lo cutoffs are defined in equations (3) and (4), using the input parameters *cutlo* and *cuthi* as thresholds to avoid adding grain boundary energy when the deviation in the order parameter from 0 or 1 is small (e.g. due to thermal fluctuations in a perfect crystal). The added potential energy U_i for atom I is given in equation (6) where it is interpolated between 0 and dE using the two threshold Ξ_i values and the ω_i value of equation (5).

The derivative of this energy expression gives the force on each atom which thus depends on the orientation of its neighbors relative to the 2 grain orientations. Only atoms near the grain boundary feel a net force which tends to drive them to one of the two grain orientations.

In equation (1), the reference vector used for each neighbor is the reference vector closest to the actual neighbor position. This means it is possible two different neighbors will use the same reference vector. In such cases, the atom in question is far from a perfect orientation and will likely receive the full dE addition, so the effect of duplicate reference vector usage is small.

The *dir* parameter determines which grain wants to grow at the expense of the other. A value of 0 means the first grain will shrink; a value of 1 means it will grow. This assumes that dE is positive. The reverse will be true if dE is negative.

The *alat* parameter is the cubic lattice constant for the fcc or bcc material and is only used to compute a cutoff distance of $1.57 * alat / \sqrt{2}$ for finding the 12 or 8 nearest neighbors of each atom (which should be valid for an fcc or bcc crystal). A longer/shorter cutoff can be imposed by adjusting *alat*. If a particular atom has less than 12 or 8 neighbors within the cutoff, the order parameter of equation (1) is effectively multiplied by 12 or 8 divided by the actual number of neighbors within the cutoff.

The *dE* parameter is the maximum amount of additional energy added to each atom in the grain which wants to shrink.

The *cutlo* and *cuthi* parameters are used to reduce the force added to bulk atoms in each grain far away from the boundary. An atom in the bulk surrounded by neighbors at the ideal grain orientation would compute an order parameter of 0 or 1 and have no force added. However, thermal vibrations in the solid will cause the order parameters to be greater than 0 or less than 1. The cutoff parameters mask this effect, allowing forces to only be added to atoms with order-parameters between the cutoff values.

File0 and *file1* are filenames for the two grains which each contain 6 vectors (6 lines with 3 values per line) which specify the grain orientations. Each vector is a displacement from a central atom (0,0,0) to a nearest neighbor atom in an fcc lattice at the proper orientation. The vector lengths should all be identical since an fcc lattice has a coordination number of 12. Only 6 are listed due to symmetry, so the list must include one from each pair of equal-and-opposite neighbors. A pair of orientation files for a Sigma=5 tilt boundary are shown below. A tutorial that can help for writing the orientation files is given in ([Wicaksono2](#))

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

No information about this fix is written to *binary restart files*.

The *fix_modify energy* option is supported by this fix to add the potential energy of atom interactions with the grain boundary driving force to the global potential energy of the system as part of *thermodynamic output*. The default setting for this fix is *fix_modify energy no*.

The *fix_modify respa* option is supported by these fixes. This allows to set at which level of the *r-RESPA* integrator a fix is adding its forces. Default is the outermost level.

This fix calculates a global scalar which can be accessed by various *output commands*. The scalar is the potential energy change due to this fix. The scalar value calculated by this fix is “extensive”.

This fix also calculates a per-atom array which can be accessed by various *output commands*. The array stores the order parameter X_i and normalized order parameter (0 to 1) for each atom. The per-atom values can be accessed on any timestep.

No parameter of this fix can be used with the *start/stop* keywords of the *run* command.

This fix is not invoked during *energy minimization*.

2.150.5 Restrictions

These fixes are part of the ORIENT package. They are only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

These fixes should only be used with fcc or bcc lattices.

2.150.6 Related commands

fix_modify

2.150.7 Default

none

(Janssens) Janssens, Olmsted, Holm, Foiles, Plimpton, Derlet, *Nature Materials*, 5, 124-127 (2006).

(Wicaksono1) Wicaksono, Sinclair, Militzer, *Computational Materials Science*, 117, 397-405 (2016).

(Wicaksono2) Wicaksono, figshare, <https://doi.org/10.6084/m9.figshare.1488628.v1> (2015).

For illustration purposes, here are example files that specify a Sigma=5 <100> tilt boundary. This is for a lattice constant of 3.5706 Angs.

file0:

```
0.798410432046075  1.7853000000000000  1.596820864092150
-0.798410432046075 1.7853000000000000 -1.596820864092150
2.395231296138225  0.0000000000000000  0.798410432046075
0.798410432046075  0.0000000000000000 -2.395231296138225
1.596820864092150  1.7853000000000000 -0.798410432046075
1.596820864092150 -1.7853000000000000 -0.798410432046075
```

file1:

```
-0.798410432046075  1.7853000000000000  1.596820864092150
0.798410432046075  1.7853000000000000 -1.596820864092150
0.798410432046075  0.0000000000000000  2.395231296138225
2.395231296138225  0.0000000000000000 -0.798410432046075
1.596820864092150  1.7853000000000000  0.798410432046075
1.596820864092150 -1.7853000000000000  0.798410432046075
```

2.151 fix orient/eco command

```
fix ID group-ID orient/eco u0 eta cutoff orientationsFile
```

- ID, group-ID are documented in fix command
- u0 = energy added to each atom (energy units)
- eta = cutoff value (usually 0.25)

- cutoff = cutoff radius for orientation parameter calculation
- orientationsFile = file that specifies orientation of each grain

2.151.1 Examples

```
fix gb all orient/eco 0.08 0.25 3.524 sigma5.ori
```

2.151.2 Description

The fix applies a synthetic driving force to a grain boundary which can be used for the investigation of grain boundary motion. The affiliation of atoms to either of the two grains forming the grain boundary is determined from an orientation-dependent order parameter as described in ([Ulomek](#)). The potential energy of atoms is either increased by an amount of $0.5*u0$ or $-0.5*u0$ according to the orientation of the surrounding crystal. This creates a potential energy gradient which pushes atoms near the grain boundary to orient according to the energetically favorable grain orientation. This fix is designed for applications in bicrystal system with one grain boundary and open ends, or two opposite grain boundaries in a periodic system. In either case, the entire system can experience a displacement during the simulation which needs to be accounted for in the evaluation of the grain boundary velocity. While the basic method is described in ([Ulomek](#)), the implementation follows the efficient implementation from ([Schratt & Mohles](#)). The synthetic potential energy added to an atom j is given by the following formulas

$$w(|\vec{r}_{jk}|) = w_{jk} = \begin{cases} \frac{|\vec{r}_{jk}|^4}{r_{\text{cut}}^4} - 2\frac{|\vec{r}_{jk}|^2}{r_{\text{cut}}^2} + 1, & |\vec{r}_{jk}| < r_{\text{cut}} \\ 0, & |\vec{r}_{jk}| \geq r_{\text{cut}} \end{cases}$$

$$\chi_j = \frac{1}{N} \sum_{l=1}^3 \left[|\psi_l^I(\vec{r}_j)|^2 - |\psi_l^{\text{II}}(\vec{r}_j)|^2 \right]$$

$$\psi_l^X(\vec{r}_j) = \sum_{k \in \square_j} w_{jk} \exp(i\vec{r}_{jk} \cdot \vec{q}_l^X)$$

$$u(\chi_j) = \frac{u_0}{2} \begin{cases} 1, & \chi_j \geq \eta \\ \sin\left(\frac{\pi\chi_j}{2\eta}\right), & -\eta < \chi_j < \eta \\ -1, & \chi_j \leq -\eta \end{cases}$$

which are fully explained in ([Ulomek](#)) and ([Schratt & Mohles](#)).

The force on each atom is the negative gradient of the synthetic potential energy. It depends on the surrounding of this atom. An atom far from the grain boundary does not experience a synthetic force as its surrounding is that of an oriented single crystal and thermal fluctuations are masked by the parameter *eta*. Near the grain boundary however, the gradient is nonzero and synthetic force terms are computed. The orientationsFile specifies the perfect oriented crystal basis vectors for the two adjoining crystals. The first three lines (line=row vector) for the energetically penalized and the last three lines for the energetically favored grain assuming $u0$ is positive. For negative $u0$, this is reversed. With the *cutoff* parameter, the size of the region around each atom which is used in the order parameter computation is defined. The cutoff must be smaller than the interaction range of the MD potential. It should at least include the nearest neighbor shell. For high temperatures or low angle grain boundaries, it might be beneficial to increase the cutoff in order to get a more precise identification of the atoms surrounding. However, computation time will increase as more atoms are considered in the order parameter and force computation. It is also worth noting that the cutoff radius must not exceed the communication distance for ghost atoms in LAMMPS. With orientationsFile, the 6 oriented crystal basis vectors are specified. Each line of the input file contains the three components of a primitive lattice vector oriented according to the grain orientation in the simulation box. The first (last) three lines correspond to the primitive lattice vectors of the first (second) grain. An example for a $\Sigma(001)$ mis-orientation is given at the end.

If no synthetic energy difference between the grains is created, $u0 = 0$, the force computation is omitted. In this case, still, the order parameter of the driving force is computed and can be used to track the grain boundary motion throughout the simulation.

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

No information about this fix is written to *binary restart files*.

The *fix_modify energy* option is supported by this fix to add the potential energy of atom interactions with the grain boundary driving force to the global potential energy of the system as part of *thermodynamic output*. The default setting for this fix is *fix_modify energy no*.

This fix calculates a per-atom array with 2 columns, which can be accessed by indices 1-1 by any command that uses per-atom values from a fix as input. See the *Howto output* doc page for an overview of LAMMPS output options.

The first column is the order parameter for each atom; the second is the thermal masking value for each atom. Both are described above.

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.

2.151.4 Restrictions

This fix is part of the ORIENT package. It is only enabled if LAMMPS was built with that package. See the *Build package* page for more info.

2.151.5 Related commands

fix_modify

fix_orient

2.151.6 Default

none

(Ulomek) Ulomek, Brien, Foiles, Mohles, Modelling Simul. Mater. Sci. Eng. 23 (2015) 025007

(Schratt & Mohles) Schratt, Mohles. Comp. Mat. Sci. 182 (2020) 109774

For illustration purposes, here is an example file that specifies a $\Sigma = 5\langle 001 \rangle$ tilt grain boundary. This is for a lattice constant of 3.52 Angstrom:

```
sigma5.ori:  
  
1.671685 0.557228 1.76212  
0.557228 -1.671685 1.76212  
2.228913 -1.114456 0.00000  
0.557228 1.671685 1.76212  
1.671685 -0.557228 1.76212  
2.228913 1.114456 0.00000
```

2.152 fix pafi command

2.152.1 Syntax

```
fix ID group-ID pafi compute-ID Temp Tdamp seed keyword values...
```

- ID, group-ID are documented in [fix](#) command
- pafi = style name of this fix command
- compute-ID = ID of a [compute property/atom](#) that holds data used by this fix
- Temp = desired temperature (temperature units)
- Tdamp = damping parameter (time units)
- seed = random number seed to use for white noise (positive integer)
- keyword = *overdamped* or *com*

overdamped value = yes or no or 1 or 0

yes or 1 = Brownian (overdamped) integration in hyperplane

no or 0 = Langevin integration in hyperplane

com value = yes or no or 1 or 0

yes or 1 = zero linear momentum, fixing center or mass (recommended)

no or 0 = do not zero linear momentum, allowing center of mass drift

2.152.2 Examples

```
compute pa all property/atom d_nx d_ny d_nz d_dnx d_dny d_dnz d_ddnx d_ddny d_ddnz
run 0 post no
fix hp all pafi pa 500.0 0.01 434 overdamped yes
```

2.152.3 Description

Perform Brownian or Langevin integration whilst constraining the system to lie in some hyperplane, which is expected to be the tangent plane to some reference pathway in a solid state system. The instantaneous value of a modified force projection is also calculated, whose time integral can be shown to be equal to the true free energy gradient along the minimum free energy path local to the reference pathway. A detailed discussion of the projection technique can be found in ([Swinburne](#)).

This fix can be used with LAMMPS as demonstrated in examples/PACKAGES/pafi, though it is primarily intended to be coupled with the PAFI C++ code, developed at <https://github.com/tomswinburne/pafi>, which distributes multiple LAMMPS workers in parallel to compute and collate hyperplane-constrained averages, allowing the calculation of free energy barriers and pathways.

A [compute property/atom](#) must be provided with 9 fields per atom coordinate, which in order are the x,y,z coordinates of a configuration on the reference path, the x,y,z coordinates of the path tangent (derivative of path position with path coordinate) and the x,y,z coordinates of the change in tangent (derivative of path tangent with path coordinate).

A 4-element vector is also calculated by this fix. The 4 components are the modified projected force, its square, the expected projection of the minimum free energy path tangent on the reference path tangent and the minimum image distance between the current configuration and the reference configuration, projected along the path tangent. This latter value should be essentially zero.

Note

When com=yes/1, which is recommended, the provided tangent vector must also have zero center of mass. This can be achieved by subtracting from each coordinate of the path tangent the average x,y,z value. The PAFI C++ code (see above) can generate these paths for use in LAMMPS.

Note

When overdamped=yes/1, the Tdamp parameter should be around 5-10 times smaller than that used in typical Langevin integration. See [fix langevin](#) for typical values.

2.152.4 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 each timestep which can be accessed by various [output commands](#).

2.152.5 Restrictions

This fix is part of the EXTRA-FIX package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

2.152.6 Default

The option defaults are com = yes, overdamped = no

(Swinburne) Swinburne and Marinica, Physical Review Letters, 120, 1 (2018)

2.153 fix pair command

2.153.1 Syntax

```
fix ID group-ID pair N pstyle name flag ...
```

- ID, group-ID are documented in [fix](#) command
- pair = style name of this fix command
- N = invoke this fix once every N timesteps
- pstyle = name of pair style to extract info from (e.g. eam)
- one or more name/flag pairs can be listed
- name = name of quantity the pair style allows extraction of
- flag = 1 if pair style needs to be triggered to produce data for name, 0 if not

2.153.2 Examples

```
fix request all pair 100 eam rho 0
fix request all pair 100 amoeba uind 0 uinp 0
```

2.153.3 Description

Added in version 15Sep2022.

Extract per-atom quantities from a pair style and store them in this fix so they can be accessed by other LAMMPS commands, e.g. by a [dump](#) command or by another [fix](#), [compute](#), or [variable](#) command.

These are example use cases:

- extract per-atom density from [pair_style eam](#) to a dump file
- extract induced dipoles from [pair_style amoeba](#) to a dump file
- extract accuracy metrics from a machine-learned potential to trigger output when a condition is met (see the [dump_modify skip](#) command)

The *N* argument determines how often the fix is invoked.

The *pstyle* argument is the name of the pair style. It can be a sub-style used in a [pair_style hybrid](#) command. If there are multiple sub-styles using the same pair style, then *pstyle* should be specified as “style:*N*”, where *N* is the number of the instance of the pair style you wish monitor (e.g., the first or second). For example, *pstyle* could be specified as “pace/extrapolation” or “amoeba” or “eam:1” or “eam:2”.

One or more *name/flag* pairs of arguments follow. Each *name* is a per-atom quantity which the pair style must recognize as an extraction request. See the doc pages for individual [pair_styles](#) to see what fix pair requests (if any) they support.

The *flag* setting determines whether this fix will also trigger the pair style to compute the named quantity so it can be extracted. If the quantity is always computed by the pair style, no trigger is needed; specify *flag* = 0. If the quantity is not always computed (e.g. because it is expensive to calculate), then specify *flag* = 1. This will trigger the quantity to be calculated only on timesteps it is needed. Again, see the doc pages for individual [pair_styles](#) to determine which fix pair requests (if any) need to be triggered with a *flag* = 1 setting.

The per-atom data extracted from the pair style is stored by this fix as either a per-atom vector or array. If there is only one *name* argument specified and the pair style computes a single value for each atom, then this fix stores it as a per-atom vector. Otherwise a per-atom array is created, with its data in the order of the *name* arguments.

For example, [pair_style amoeba](#) allows extraction of two named quantities: “uind” and “uinp”, both of which are 3-vectors for each atom, i.e. dipole moments. In the example below a 6-column per-atom array will be created. Columns 1-3 will store the “uind” values; columns 4-6 will store the “uinp” values.

```
pair_style amoeba
fix ex all pair 10 amoeba uind 0 uinp 0
```

2.153.4 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.

As explained above, this fix produces a per-atom vector or array which can be accessed by various *output commands*. If an array is produced, the number of columns is the sum of the number of per-atom quantities produced by each *name* argument requested from the pair style.

2.153.5 Restrictions

none

2.153.6 Related commands

compute pair

2.153.7 Default

none

2.154 fix phonon command

2.154.1 Syntax

```
fix ID group-ID phonon N Noutput Nwait map_file prefix keyword values ...
```

- ID, group-ID are documented in *fix* command
- phonon = style name of this fix command
- N = measure the Green's function every this many timesteps
- Noutput = output the dynamical matrix every this many measurements
- Nwait = wait this many timesteps before measuring
- map_file = *file* or *GAMMA*

file is the file that contains the mapping info between atom ID and the lattice indices.

GAMMA flags to treat the whole simulation box as a unit cell, so that the mapping info can be generated internally. In this case, dynamical matrix at only the gamma-point will/can be evaluated.

- prefix = prefix for output files
- one or none keyword/value pairs may be appended
- keyword = *sysdim* or *nasr*

sysdim value = *d*

d = dimension of the system, usually the same as the MD model dimension

nasr value = *n*

n = number of iterations to enforce the acoustic sum rule

2.154.2 Examples

```
fix 1 all phonon 20 5000 200000 map.in LJ1D sysdim 1
fix 1 all phonon 20 5000 200000 map.in EAM3D
fix 1 all phonon 10 5000 500000 GAMMA EAM0D nasr 100
```

2.154.3 Description

Calculate the dynamical matrix from molecular dynamics simulations based on fluctuation-dissipation theory for a group of atoms.

Consider a crystal with N unit cells in three dimensions labeled $l = (l_1, l_2, l_3)$ where l_i are integers. Each unit cell is defined by three linearly independent vectors $\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3$ forming a parallelepiped, containing K basis atoms labeled k .

Based on fluctuation-dissipation theory, the force constant coefficients of the system in reciprocal space are given by ([Campana , Kong](#))

$$\mathbf{\Gamma}_{k\alpha,k'\beta}(\mathbf{q}) = k_B T \mathbf{G}_{k\alpha,k'\beta}^{-1}(\mathbf{q})$$

where \mathbf{G} is the Green's functions coefficients given by

$$\mathbf{G}_{k\alpha,k'\beta}(\mathbf{q}) = \langle \mathbf{u}_{k\alpha}(\mathbf{q}) \bullet \mathbf{u}_{k'\beta}^*(\mathbf{q}) \rangle$$

where $\langle \dots \rangle$ denotes the ensemble average, and

$$\mathbf{u}_{k\alpha}(\mathbf{q}) = \sum_l \mathbf{u}_{lk\alpha} \exp(i\mathbf{qr}_l)$$

is the α component of the atomic displacement for the k th atom in the unit cell in reciprocal space at \mathbf{q} . In practice, the Green's functions coefficients can also be measured according to the following formula,

$$\mathbf{G}_{k\alpha,k'\beta}(\mathbf{q}) = \langle \mathbf{R}_{k\alpha}(\mathbf{q}) \bullet \mathbf{R}_{k'\beta}^*(\mathbf{q}) \rangle - \langle \mathbf{R} \rangle_{k\alpha}(\mathbf{q}) \bullet \langle \mathbf{R} \rangle_{k'\beta}^*(\mathbf{q})$$

where \mathbf{R} is the instantaneous positions of atoms, and $\langle \mathbf{R} \rangle$ is the averaged atomic positions. It gives essentially the same results as the displacement method and is easier to implement in an MD code.

Once the force constant matrix is known, the dynamical matrix \mathbf{D} can then be obtained by

$$\mathbf{D}_{k\alpha,k'\beta}(\mathbf{q}) = (m_k m_{k'})^{-\frac{1}{2}} \mathbf{\Gamma}_{k\alpha,k'\beta}(\mathbf{q})$$

whose eigenvalues are exactly the phonon frequencies at \mathbf{q} .

This fix uses positions of atoms in the specified group and calculates two-point correlations. To achieve this. the positions of the atoms are examined every N_{every} steps and are Fourier-transformed into reciprocal space, where the averaging process and correlation computation is then done. After every N_{output} measurements, the matrix $\mathbf{G}(\mathbf{q})$ is calculated and inverted to obtain the elastic stiffness coefficients. The dynamical matrices are then constructed and written to *prefix.bin.timestep* files in binary format and to the file *prefix.log* for each wave-vector \mathbf{q} .

A detailed description of this method can be found in ([Kong2011](#)).

The *sysdim* keyword is optional. If specified with a value smaller than the dimensionality of the LAMMPS simulation, its value is used for the dynamical matrix calculation. For example, using LAMMPS to model a 2D or 3D system, the phonon dispersion of a 1D atomic chain can be computed using *sysdim* = 1.

The *nasr* keyword is optional. An iterative procedure is employed to enforce the acoustic sum rule on Φ at Γ , and the number provided by keyword *nasr* gives the total number of iterations. For a system whose unit cell has only one atom, *nasr* = 1 is sufficient; for other systems, *nasr* = 10 is typically sufficient.

The *map_file* contains the mapping information between the lattice indices and the atom IDs, which tells the code which atom sits at which lattice point; the lattice indices start from 0. An auxiliary code, *latgen*, can be employed to generate the compatible map file for various crystals.

In case one simulates a non-periodic system, where the whole simulation box is treated as a unit cell, one can set *map_file* as *GAMMA*, so that the mapping info will be generated internally and a file is not needed. In this case, the dynamical matrix at only the gamma-point will/can be evaluated. Please keep in mind that fix-phonon is designed for crystals, it will be inefficient and even degrade the performance of LAMMPS in case the unit cell is too large.

The calculated dynamical matrix elements are written out in *energy/distance^2/mass* units. The coordinates for *q* points in the log file is in the units of the basis vectors of the corresponding reciprocal lattice.

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

No information about this fix is written to *binary restart files*.

The *fix_modify temp* option is supported by this fix. You can use it to change the temperature compute from *thermo_temp* to the one that reflects the true temperature of atoms in the group.

No global scalar or vector or per-atom quantities are stored by this fix for access by various *output commands*.

Instead, this fix outputs its initialization information (including mapping information) and the calculated dynamical matrices to the file *prefix.log*, with the specified *prefix*. The dynamical matrices are also written to files *prefix.bin.timestep* in binary format. These can be read by the post-processing tool in tools/phonon to compute the phonon density of states and/or phonon dispersion curves.

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*.

2.154.5 Restrictions

This fix assumes a crystalline system with periodical lattice. The temperature of the system should not exceed the melting temperature to keep the system in its solid state.

This fix is part of the PHONON package. It is only enabled if LAMMPS was built with that package. This fix also requires LAMMPS to be built with 3d-FFT support which is included in the KSPACE package. See the *Build package* page for more info.

2.154.6 Related commands

compute msd, dynamical_matrix

2.154.7 Default

The option defaults are sysdim = the same dimension as specified by the *dimension* command, and nasr = 20.

(Campana) C. Campana and M. H. Muser, *Practical Green's function approach to the simulation of elastic semi-infinite solids*, Phys. Rev. B [74], 075420 (2006)

(Kong) L.T. Kong, G. Bartels, C. Campana, C. Denniston, and Martin H. Muser, *Implementation of Green's function molecular dynamics: An extension to LAMMPS*, Computer Physics Communications [180](6):1004-1010 (2009).

L.T. Kong, C. Denniston, and Martin H. Muser, *An improved version of the Green's function molecular dynamics method*, Computer Physics Communications [182](2):540-541 (2011).

(Kong2011) L.T. Kong, *Phonon dispersion measured directly from molecular dynamics simulations*, Computer Physics Communications [182](10):2201-2207, (2011).

2.155 fix pimd/langevin command

2.156 fix pimd/nvt command

2.156.1 Syntax

`fix ID group-ID style keyword value ...`

- ID, group-ID are documented in [fix](#) command
- style = *pimd/langevin* or *pimd/nvt* = style name of this fix command
- zero or more keyword/value pairs may be appended
- keywords for style *pimd/nvt*
 - keywords = method or fmass or sp or temp or nhc
method value = pimd or nmpimd or cmd
fmass value = scaling factor on mass
sp value = scaling factor on Planck constant
temp value = temperature (temperature units)
nhc value = Nc = number of chains in Nose-Hoover thermostat
- keywords for style *pimd/langevin*
 - keywords = method or integrator or ensemble or fmmode or fmass or scale or temp or thermostat
method value = nmpimd (default) or pimd
integrator value = obabo or baoab
fmmode value = physical or normal
fmass value = scaling factor on mass
temp value = temperature (temperature unit)
temperature = target temperature of the thermostat
thermostat values = style seed
style value = PILE_L
seed = random number generator seed
tau value = thermostat damping parameter (time unit)
scale value = scaling factor of the damping times of non-centroid modes of PILE_L thermostat
iso or aniso values = pressure (pressure unit)
pressure = scalar external pressure of the barostat
barostat value = BZP or MTTK
taup value = barostat damping parameter (time unit)
fixcom value = yes or no
lj values = epsilon sigma mass planck mvv2e
epsilon = energy scale for reduced units (energy units)
sigma = length scale for reduced units (length units)
mass = mass scale for reduced units (mass units)
planck = Planck's constant for other unit style

`mvv2e` = mass * velocity² to energy conversion factor for other unit style

2.156.2 Examples

```
fix 1 all pimd/nvt method nmpimd fmass 1.0 sp 2.0 temp 300.0 nhc 4
fix 1 all pimd/langevin ensemble npt integrator obabo temp 113.15 thermostat PILE_L 1234 tau 1.0 iso 1.
  ↴0 barostat BZP taup 1.0
```

2.156.3 Description

Changed in version 28Mar2023.

Fix `pimd` was renamed to fix `pimd/nvt` and fix `pimd/langevin` was added.

These fix commands perform quantum molecular dynamics simulations based on the Feynman path-integral to include effects of tunneling and zero-point motion. In this formalism, the isomorphism of a quantum partition function for the original system to a classical partition function for a ring-polymer system is exploited, to efficiently sample configurations from the canonical ensemble ([Feynman](#)).

The classical partition function and its components are given by the following equations:

$$Z = \int d\mathbf{q}d\mathbf{p} \cdot \exp[-\beta H_{eff}]$$

$$H_{eff} = \left(\sum_{i=1}^P \frac{p_i^2}{2M_i} \right) + V_{eff}$$

$$V_{eff} = \sum_{i=1}^P \left[\frac{mP}{2\beta^2\hbar^2} (q_i - q_{i+1})^2 + \frac{1}{P} V(q_i) \right]$$

M_i is the fictitious mass of the i -th mode, and m is the actual mass of the atoms.

The interested user is referred to any of the numerous references on this methodology, but briefly, each quantum particle in a path integral simulation is represented by a ring-polymer of P quasi-beads, labeled from 1 to P . During the simulation, each quasi-bead interacts with beads on the other ring-polymers with the same imaginary time index (the second term in the effective potential above). The quasi-beads also interact with the two neighboring quasi-beads through the spring potential in imaginary-time space (first term in effective potential). To sample the canonical ensemble, any thermostat can be applied.

Fix `pimd/nvt` applies a Nose-Hoover massive chain thermostat ([Tuckerman](#)). With the massive chain algorithm, a chain of NH thermostats is coupled to each degree of freedom for each quasi-bead. The keyword `temp` sets the target temperature for the system and the keyword `nhc` sets the number Nc of thermostats in each chain. For example, for a simulation of N particles with P beads in each ring-polymer, the total number of NH thermostats would be $3 \times N \times P \times Nc$.

Fix `pimd/langevin` implements a Langevin thermostat in the normal mode representation, and also provides a barostat to sample the NPH/NPT ensembles.

Note

Both these `fix` styles implement a complete velocity-verlet integrator combined with a thermostat, so no other time integration fix should be used.

The `method` keyword determines what style of PIMD is performed. A value of `pimd` is standard PIMD. A value of `nmpimd` is for normal-mode PIMD. A value of `cmd` is for centroid molecular dynamics (CMD). The difference between the styles is as follows.

In standard PIMD, the value used for a bead's fictitious mass is arbitrary. A common choice is to use $M_i = m/P$, which results in the mass of the entire ring-polymer being equal to the real quantum particle. But it can be difficult to efficiently integrate the equations of motion for the stiff harmonic interactions in the ring polymers.

A useful way to resolve this issue is to integrate the equations of motion in a normal mode representation, using Normal Mode Path-Integral Molecular Dynamics (NMPIMD) ([Cao1](#)). In NMPIMD, the NH chains are attached to each normal mode of the ring-polymer and the fictitious mass of each mode is chosen as $M_k = \text{eigenvalue of the } k\text{-th normal mode}$ for $k > 0$. The $k = 0$ mode, referred to as the zero-frequency mode or centroid, corresponds to overall translation of the ring-polymer and is assigned the mass of the real particle.

Note

Motion of the centroid can be effectively uncoupled from the other normal modes by scaling the fictitious masses to achieve a partial adiabatic separation. This is called a Centroid Molecular Dynamics (CMD) approximation ([Cao2](#)). The time-evolution (and resulting dynamics) of the quantum particles can be used to obtain centroid time correlation functions, which can be further used to obtain the true quantum correlation function for the original system. The CMD method also uses normal modes to evolve the system, except only the $k > 0$ modes are thermostatted, not the centroid degrees of freedom.

Added in version 21Nov2023: Mode *pimd* added to fix pimd/langevin.

Fix pimd/langevin supports the *method* values *nmpimd* and *pimd*. The default value is *nmpimd*. If *method* is *nmpimd*, the normal mode representation is used to integrate the equations of motion. The exact solution of harmonic oscillator is used to propagate the free ring polymer part of the Hamiltonian. If *method* is *pimd*, the Cartesian representation is used to integrate the equations of motion. The harmonic force is added to the total force of the system, and the numerical integrator is used to propagate the Hamiltonian.

The keyword *integrator* specifies the Trotter splitting method used by *fix pimd/langevin*. See ([Liu](#)) for a discussion on the OBABO and BAOAB splitting schemes. Typically either of the two should work fine.

The keyword *fmass* sets a further scaling factor for the fictitious masses of beads, which can be used for the Partial Adiabatic CMD ([Hone](#)), or to be set as *P*, which results in the fictitious masses to be equal to the real particle masses.

The keyword *fmode* of *fix pimd/langevin* determines the mode of fictitious mass preconditioning. There are two options: *physical* and *normal*. If *fmode* is *physical*, then the physical mass of the particles are used (and then multiplied by *fmass*). If *fmode* is *normal*, then the physical mass is first multiplied by the eigenvalue of each normal mode, and then multiplied by *fmass*. More precisely, the fictitious mass of *fix pimd/langevin* is determined by two factors: *fmode* and *fmass*. If *fmode* is *physical*, then the fictitious mass is

$$M_i = \text{fmass} \times m$$

If *fmode* is *normal*, then the fictitious mass is

$$M_i = \text{fmass} \times \lambda_i \times m$$

where λ_i is the eigenvalue of the i -th normal mode.

Note

Fictitious mass is only used in the momentum of the equation of motion ($\mathbf{p}_i = M_i \mathbf{v}_i$), and not used in the spring elastic energy ($\sum_{i=1}^P \frac{1}{2} m \omega_p^2 (q_i - q_{i+1})^2$, m is always the actual mass of the particles).

The keyword *sp* is a scaling factor on Planck's constant, which can be useful for debugging or other purposes. The default value of 1.0 is appropriate for most situations.

The keyword *ensemble* for fix style *pimd/langevin* determines which ensemble is it going to sample. The value can be *nve* (microcanonical), *nvt* (canonical), *nph* (isoenthalpic), and *npt* (isothermal-isobaric).

The keyword *temp* specifies temperature parameter for fix styles *pimd/nvt* and *pimd/langevin*. It should read a positive floating-point number.

Note

For pimd simulations, a temperature values should be specified even for nve ensemble. Temperature will make a difference for nve pimd, since the spring elastic frequency between the beads will be affected by the temperature.

The keyword *thermostat* reads *style* and *seed* of thermostat for fix style *pimd/langevin*. *style* can only be *PILE_L* (path integral Langevin equation local thermostat, as described in [Ceriotti](#)), and *seed* should a positive integer number, which serves as the seed of the pseudo random number generator.

Note

The fix style *pimd/langevin* uses the stochastic PILE L thermostat to control temperature. This thermostat works on the normal modes of the ring polymer. The *tau* parameter controls the centroid mode, and the *scale* parameter controls the non-centroid modes.

The keyword *tau* specifies the thermostat damping time parameter for fix style *pimd/langevin*. It is in time unit. It only works on the centroid mode.

The keyword *scale* specifies a scaling parameter for the damping times of the non-centroid modes for fix style *pimd/langevin*. The default damping time of the non-centroid mode *i* is $\frac{P}{\beta\hbar}\sqrt{\lambda_i \times \text{fmass}}$ (*fmmode* is *physical*) or $\frac{P}{\beta\hbar}\sqrt{\text{fmass}}$ (*fmmode* is *normal*). The damping times of all non-centroid modes are the default values divided by *scale*.

The barostat parameters for fix style *pimd/langevin* with *npt* or *nph* ensemble is specified using one of *iso* and *aniso* keywords. A *pressure* value should be given with pressure unit. The keyword *iso* means couple all 3 diagonal components together when pressure is computed (hydrostatic pressure), and dilate/contract the dimensions together. 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.

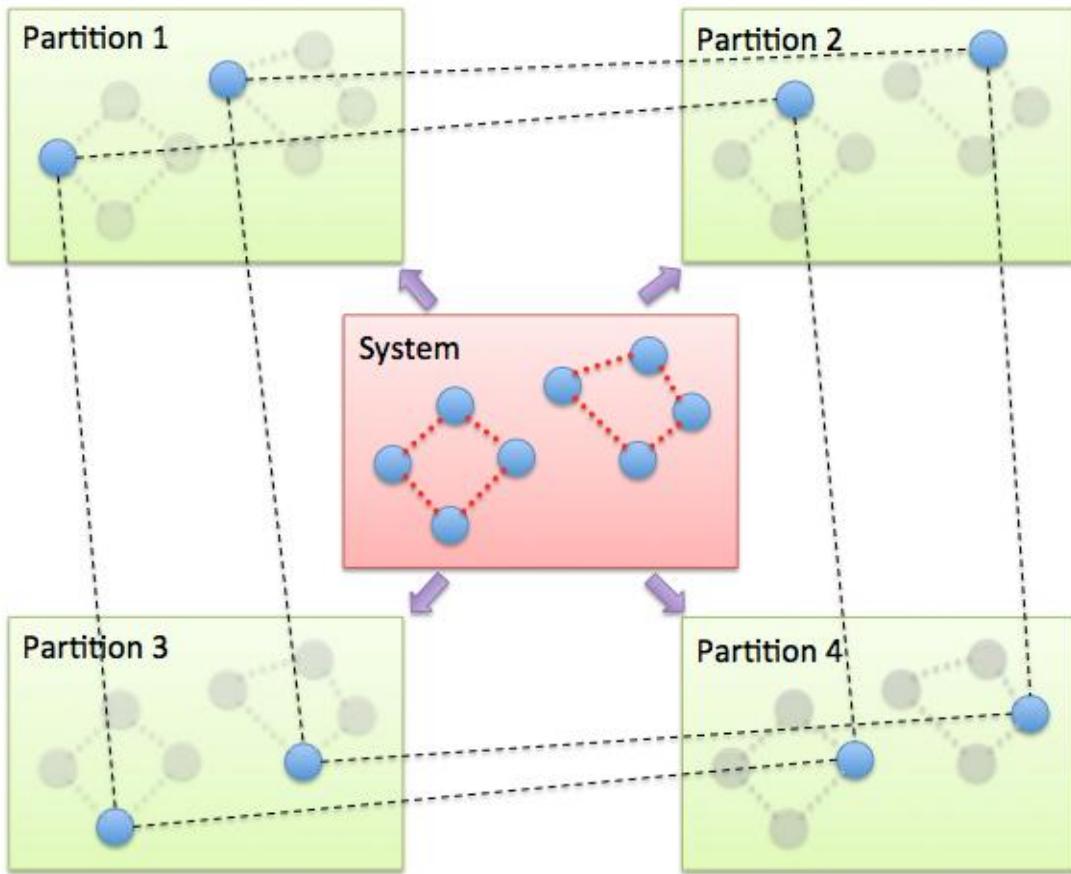
The keyword *barostat* reads *style* of barostat for fix style *pimd/langevin*. *style* can be *BZP* (Bussi-Zykova-Parrinello, as described in [Bussi](#)) or *MTTK* (Martyna-Tuckerman-Tobias-Klein, as described in [Martyna1](#) and [Martyna2](#)).

The keyword *taup* specifies the barostat damping time parameter for fix style *pimd/langevin*. It is in time unit.

The keyword *fixcom* specifies whether the center-of-mass of the extended ring-polymer system is fixed during the pimd simulation. Once *fixcom* is set to be *yes*, the center-of-mass velocity will be distracted from the centroid-mode velocities in each step.

The keyword *lj* should be used if [lj units](#) is used for fix *pimd/langevin*. Typically one may want to use reduced units to run the simulation, and then convert the results into some physical units (for example, [metal units](#)). In this case, the 5 quantities in the physical mass units are needed: epsilon (energy scale), sigma (length scale), mass, Planck's constant, mvv2e (mass * velocity^2 to energy conversion factor). Planck's constant and mvv2e can be found in src/update.cpp. If there is no need to convert reduced units to physical units, you can omit the keyword *lj* and these five values will be set to 1.

The PIMD algorithm in LAMMPS is implemented as a hyper-parallel scheme as described in [Calhoun](#). In LAMMPS this is done by using [multi-replica feature](#) in LAMMPS, where each quasi-particle system is stored and simulated on a separate partition of processors. The following diagram illustrates this approach. The original system with 2 ring polymers is shown in red. Since each ring has 4 quasi-beads (imaginary time slices), there are 4 replicas of the system, each running on one of the 4 partitions of processors. Each replica (shown in green) owns one quasi-bead in each ring.



To run a PIMD simulation with M quasi-beads in each ring polymer using N MPI tasks for each partition's domain-decomposition, you would use $P = M \times N$ processors (cores) and run the simulation as follows:

```
mpirun -np P lmp_mpi -partition MxN -in script
```

Note that in the LAMMPS input script for a multi-partition simulation, it is often very useful to define a *uloop-style variable* such as

```
variable ibead uloop M pad
```

where M is the number of quasi-beads (partitions) used in the calculation. The uloop variable can then be used to manage I/O related tasks for each of the partitions, e.g.

```
dump dcd all dcd 10 system_${ibead}.dcd
dump 1 all custom 100 ${ibead}.xyz id type x y z vx vy vz ix iy iz fx fy fz
restart 1000 system_${ibead}.restart1 system_${ibead}.restart2
read_restart system_${ibead}.restart2
```

Note

Fix *pimd/langevin* dumps the Cartesian coordinates, but dumps the velocities and forces in the normal mode representation. If the Cartesian velocities and forces are needed, it is easy to perform the transformation when doing post-processing.

It is recommended to dump the image flags (*ix iy iz*) for fix *pimd/langevin*. It will be useful if you want to calculate some estimators during post-processing.

Major differences of *fix pimd/nvt* and *fix pimd/langevin* are:

1. *Fix pimd/nvt* includes Cartesian pimd, normal mode pimd, and centroid md. *Fix pimd/langevin* only intends to support normal mode pimd, as it is commonly enough for thermodynamic sampling.
2. *Fix pimd/nvt* uses Nose-Hoover chain thermostat. *Fix pimd/langevin* uses Langevin thermostat.
3. *Fix pimd/langevin* provides barostat, so the npt ensemble can be sampled. *Fix pimd/nvt* only support nvt ensemble.
4. *Fix pimd/langevin* provides several quantum estimators in output.
5. *Fix pimd/langevin* allows multiple processes for each bead. For *fix pimd/nvt*, there is a large chance that multi-process tasks for each bead may fail.
6. The dump of *fix pimd/nvt* are all Cartesian. *Fix pimd/langevin* dumps normal-mode velocities and forces, and Cartesian coordinates.

Initially, the inter-replica communication and normal mode transformation parts of *fix pimd/langevin* are written based on those of *fix pimd/nvt*, but are significantly revised.

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

Fix pimd/nvt writes the state of the Nose/Hoover thermostat over all quasi-beads to *binary restart files*. See the [read_restart](#) command for info on how to re-specify a fix in an input script that reads a restart file, so that the operation of the fix continues in an uninterrupted fashion.

Fix pimd/langevin writes the state of the barostat overall beads to *binary restart files*. Since it uses a stochastic thermostat, the state of the thermostat is not written. However, the state of the system can be restored by reading the restart file, except that it will re-initialize the random number generator.

None of the [fix_modify](#) options are relevant to fix pimd/nvt.

Fix pimd/nvt computes a global 3-vector, which can be accessed by various [output commands](#). The three quantities in the global vector are:

1. the total spring energy of the quasi-beads,
2. the current temperature of the classical system of ring polymers,
3. the current value of the scalar virial estimator for the kinetic energy of the quantum system ([Herman](#)).

The vector values calculated by *fix pimd/nvt* are “extensive”, except for the temperature, which is “intensive”.

Fix pimd/langevin computes a global vector of quantities, which can be accessed by various [output commands](#). Note that it outputs multiple log files, and different log files contain information about different beads or modes (see detailed explanations below). If *ensemble* is *nve* or *nvt*, the vector has 10 values:

1. kinetic energy of the normal mode
2. spring elastic energy of the normal mode
3. potential energy of the bead
4. total energy of all beads (conserved if *ensemble* is *nve*)
5. primitive kinetic energy estimator
6. virial energy estimator

7. centroid-virial energy estimator
8. primitive pressure estimator
9. thermodynamic pressure estimator
10. centroid-virial pressure estimator

The first 3 are different for different log files, and the others are the same for different log files.

If *ensemble* is *nph* or *npt*, the vector stores internal variables of the barostat. If *iso* is used, the vector has 15 values:

1. kinetic energy of the normal mode
2. spring elastic energy of the normal mode
3. potential energy of the bead
4. total energy of all beads (conserved if *ensemble* is *nve*)
5. primitive kinetic energy estimator
6. virial energy estimator
7. centroid-virial energy estimator
8. primitive pressure estimator
9. thermodynamic pressure estimator
10. centroid-virial pressure estimator
11. barostat velocity
12. barostat kinetic energy
13. barostat potential energy
14. barostat cell Jacobian
15. enthalpy of the extended system (sum of 4, 12, 13, and 14; conserved if *ensemble* is *nph*)

If *aniso* or *x* or *y* or *z* is used for the barostat, the vector has 17 values:

1. kinetic energy of the normal mode
2. spring elastic energy of the normal mode
3. potential energy of the bead
4. total energy of all beads (conserved if *ensemble* is *nve*)
5. primitive kinetic energy estimator
6. virial energy estimator
7. centroid-virial energy estimator
8. primitive pressure estimator
9. thermodynamic pressure estimator
10. centroid-virial pressure estimator
11. x component of barostat velocity
12. y component of barostat velocity
13. z component of barostat velocity
14. barostat kinetic energy

15. barostat potential energy
16. barostat cell Jacobian
17. enthalpy of the extended system (sum of 4, 14, 15, and 16; conserved if *ensemble* is *nph*)

No parameter of fix *pimd/nvt* or *pimd/langevin* can be used with the *start/stop* keywords of the *run* command. Fix *pimd/nvt* or *pimd/langevin* is not invoked during *energy minimization*.

2.156.5 Restrictions

These fixes are part of the REPLICA package. They are only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

Fix *pimd/nvt* cannot be used with *lj units*. Fix *pimd/langevin* can be used with *lj units*. See the above part for how to use it.

A PIMD simulation can be initialized with a single data file read via the *read_data* command. However, this means all quasi-beads in a ring polymer will have identical positions and velocities, resulting in identical trajectories for all quasi-beads. To avoid this, users can simply initialize velocities with different random number seeds assigned to each partition, as defined by the *uloop* variable, e.g.

```
velocity all create 300.0 1234${ibead} rot yes dist gaussian
```

2.156.6 Default

The keyword defaults for fix *pimd/nvt* are method = pimd, fmass = 1.0, sp = 1.0, temp = 300.0, and nhc = 2.

(Feynman) R. Feynman and A. Hibbs, Chapter 7, Quantum Mechanics and Path Integrals, McGraw-Hill, New York (1965).

(Tuckerman) M. Tuckerman and B. Berne, J Chem Phys, 99, 2796 (1993).

(Cao1) J. Cao and B. Berne, J Chem Phys, 99, 2902 (1993).

(Cao2) J. Cao and G. Voth, J Chem Phys, 100, 5093 (1994).

(Hone) T. Hone, P. Rossky, G. Voth, J Chem Phys, 124, 154103 (2006).

(Calhoun) A. Calhoun, M. Pavese, G. Voth, Chem Phys Letters, 262, 415 (1996).

(Herman) M. F. Herman, E. J. Bruskin, B. J. Berne, J Chem Phys, 76, 5150 (1982).

(Bussi) G. Bussi, T. Zykova-Timan, M. Parrinello, J Chem Phys, 130, 074101 (2009).

(Ceriotti) M. Ceriotti, M. Parrinello, T. Markland, D. Manolopoulos, J. Chem. Phys. 133, 124104 (2010).

(Martyna1) G. Martyna, D. Tobias, M. Klein, J. Chem. Phys. 101, 4177 (1994).

(Martyna2) G. Martyna, A. Hughes, M. Tuckerman, J. Chem. Phys. 110, 3275 (1999).

(Liu) J. Liu, D. Li, X. Liu, J. Chem. Phys. 145, 024103 (2016).

2.157 fix planeforce command

2.157.1 Syntax

```
fix ID group-ID planeforce x y z
```

- ID, group-ID are documented in [fix](#) command
- planeforce = style name of this fix command
- x y z = 3-vector that is normal to the plane

2.157.2 Examples

```
fix hold boundary planeforce 1.0 0.0 0.0
```

2.157.3 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.

2.157.4 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.

2.157.5 Restrictions

none

2.157.6 Related commands

[fix lineforce](#)

2.157.7 Default

none

2.158 fix plumed command

2.158.1 Syntax

```
fix ID group-ID plumed keyword value ...
```

- ID, group-ID are documented in [fix](#) command
- plumed = style name of this fix command
- keyword = *plumedfile* or *outfile*
plumedfile arg = name of PLUMED input file to use (default: NULL)
outfile arg = name of file on which to write the PLUMED log (default: NULL)

2.158.2 Examples

```
fix pl all plumed plumedfile plumed.dat outfile p.log
```

2.158.3 Description

This fix instructs LAMMPS to call the [PLUMED](#) library, which allows one to perform various forms of trajectory analysis on the fly and to also use methods such as umbrella sampling and metadynamics to enhance the sampling of phase space.

The documentation included here only describes the fix plumed command itself. This command is LAMMPS specific, whereas most of the functionality implemented in PLUMED will work with a range of MD codes, and when PLUMED is used as a stand alone code for analysis. The full [documentation for PLUMED](#) is available online and included in the PLUMED source code. The PLUMED library development is hosted at <https://github.com/plumed/plumed2>. A detailed discussion of the code can be found in ([Tribello](#)).

There is an example input for using this package with LAMMPS in the examples/PACKAGES/plumed directory.

The command to make LAMMPS call PLUMED during a run requires two keyword value pairs pointing to the PLUMED input file and an output file for the PLUMED log. The user must specify these arguments every time PLUMED is to be used. Furthermore, the fix plumed command should appear in the LAMMPS input file **after** relevant input parameters (e.g. the timestep) have been set.

The *group-ID* entry is ignored. LAMMPS will always pass all the atoms to PLUMED and there can only be one instance of the plumed fix at a time. The way the plumed fix is implemented ensures that the minimum amount of information required is communicated. Furthermore, PLUMED supports multiple, completely independent collective variables, multiple independent biases and multiple independent forms of analysis. There is thus really no restriction in functionality by only allowing only one plumed fix in the LAMMPS input.

The *plumedfile* keyword allows the user to specify the name of the PLUMED input file. Instructions as to what should be included in a plumed input file can be found in the [documentation for PLUMED](#)

The *outfile* keyword allows the user to specify the name of a file in which to output the PLUMED log. This log file normally just repeats the information that is contained in the input file to confirm it was correctly read and parsed. The names of the files in which the results are stored from the various analysis options performed by PLUMED will be specified by the user in the PLUMED input file.

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

When performing a restart of a calculation that involves PLUMED you must include a RESTART command in the PLUMED input file as detailed in the [PLUMED documentation](#). When the restart command is found in the PLUMED input PLUMED will append to the files that were generated in the run that was performed previously. No part of the PLUMED restart data is included in the LAMMPS restart files. Furthermore, any history dependent bias potentials that were accumulated in previous calculations will be read in when the RESTART command is included in the PLUMED input.

The *fix_modify energy* option is supported by this fix to add the energy change from the biasing force added by PLUMED to the global potential energy of the system as part of *thermodynamic output*. The default setting for this fix is *fix_modify energy yes*.

The *fix_modify virial* option is supported by this fix to add the contribution from the biasing force to the global pressure of the system via the *compute pressure* command. This can be accessed by *thermodynamic output*. The default setting for this fix is *fix_modify virial yes*.

This fix computes a global scalar which can be accessed by various *output commands*. The scalar is the PLUMED energy mentioned above. The scalar value calculated by this fix is “extensive”.

Note that other quantities of interest can be output by commands that are native to PLUMED.

2.158.5 Restrictions

This fix is part of the PLUMED package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

There can only be one fix plumed command active at a time.

2.158.6 Related commands

fix smd *fix colvars*

2.158.7 Default

The default options are plumedfile = NULL and outfile = NULL

(**Tribello**) G.A. Tribello, M. Bonomi, D. Branduardi, C. Camilloni and G. Bussi, Comp. Phys. Comm 185, 604 (2014)

2.159 fix poems command

2.159.1 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

2.159.2 Examples

```
fix 3 fluid poems group clump1 clump2 clump3
fix 3 fluid poems file cluster.list
```

2.159.3 Description

Treats one or more sets of atoms as coupled rigid bodies. This means that each timestep the total force and torque on each rigid body is computed and the coordinates and velocities of the atoms are updated so that the collection of bodies move as a coupled set. This can be useful for treating a large biomolecule as a collection of connected, coarse-grained particles.

The coupling, associated motion constraints, and time integration is performed by the software package *Parallelizable Open source Efficient Multibody Software (POEMS)* which computes the constrained rigid-body motion of articulated (jointed) multibody systems ([Anderson](#)). POEMS was written and is distributed by Prof Kurt Anderson, his graduate student Rudranarayan Mukherjee, and other members of his group at Rensselaer Polytechnic Institute (RPI). Rudranarayan developed the LAMMPS/POEMS interface. For copyright information on POEMS and other details, please refer to the documents in the poems directory distributed with LAMMPS.

This fix updates the positions and velocities of the rigid atoms with a constant-energy time integration, so you should not update the same atoms via other fixes (e.g. nve, nvt, npt, temp/rescale, langevin).

Each body must have a non-degenerate inertia tensor, which means it must contain at least 3 non-collinear atoms. Which atoms are in which bodies can be defined via several options.

For option *group*, each of the listed groups is treated as a rigid body. Note that only atoms that are also in the fix group are included in each rigid body.

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

For option *file*, sets of atoms are read from the specified file and each set is treated as a rigid body. Each line of the file specifies a rigid body in the following format:

ID type atom1-ID atom2-ID atom3-ID ...

ID as an integer from 1 to M (the number of rigid bodies). Type is any integer; it is not used by the fix poems command. The remaining arguments are IDs of atoms in the rigid body, each typically from 1 to N (the number of atoms in the system). Only atoms that are also in the fix group are included in each rigid body. Blank lines and lines that begin with '#' are skipped.

A connection between a pair of rigid bodies is inferred if one atom is common to both bodies. The POEMS solver treats that atom as a spherical joint with 3 degrees of freedom. Currently, a collection of bodies can only be connected by joints as a linear chain. The entire collection of rigid bodies can represent one or more chains. Other connection topologies (tree, ring) are not allowed, but will be added later. Note that if no joints exist, it is more efficient to use the [fix rigid](#) command to simulate the system.

When the poems fix is defined, it will print out statistics on the total # of clusters, bodies, joints, atoms involved. A cluster in this context means a set of rigid bodies connected by joints.

For computational efficiency, you should turn off pairwise and bond interactions within each rigid body, as they no longer contribute to the motion. The "neigh_modify exclude" and "delete_bonds" commands can be used to do this if each rigid body is a group.

For computational efficiency, you should only define one fix poems which includes all the desired rigid bodies. LAMMPS will allow multiple poems fixes to be defined, but it is more expensive.

The degrees-of-freedom removed by coupled rigid bodies are accounted for in temperature and pressure computations. Similarly, the rigid body contribution to the pressure virial is also accounted for. The latter is only correct if forces within the bodies have been turned off, and there is only a single fix poems defined.

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

No information about this fix is written to *binary restart files*.

The *fix_modify virial* option is supported by this fix to add the contribution due to the added forces and torques on atoms to both the global pressure and per-atom stress of the system via the *compute pressure* and *compute stress/atom* commands. The former can be accessed by *thermodynamic output*. The default setting for this fix is *fix_modify virial yes*.

The *fix_modify bodyforces* option is supported by this fix style to set whether per-body forces and torques are computed early or late in a timestep, i.e. at the post-force stage or at the final-integrate stage, respectively.

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*.

2.159.5 Restrictions

This fix is part of the *POEMS* package. It is only enabled if LAMMPS was built with that package, which also requires the POEMS library be built and linked with LAMMPS. See the *Build package* page for more info.

2.159.6 Related commands

fix rigid, delete_bonds, neigh_modify exclude

2.159.7 Default

none

(Anderson) Anderson, Mukherjee, Critchley, Ziegler, and Lipton “POEMS: Parallelizable Open-source Efficient Multibody Software “, Engineering With Computers (2006). ([link to paper](#))

2.160 fix polarize/bem/gmres command

2.161 fix polarize/bem/icc command

2.162 fix polarize/functional command

2.162.1 Syntax

`fix ID group-ID style nevery tolerance`

- ID, group-ID are documented in [fix](#) command
- style = *polarize/bem/gmres* or *polarizer/bem/icc* or *polarize/functional*
- nevery = this fixed is invoked every this many timesteps
- tolerance = the relative tolerance for the iterative solver to stop

2.162.2 Examples

```
fix 2 interface polarize/bem/gmres 5 0.0001
fix 1 interface polarize/bem/icc 1 0.0001
fix 3 interface polarize/functional 1 0.0001
```

Used in input scripts:

```
examples/PACKAGES/dielectric/in.confined
examples/PACKAGES/dielectric/in.nopbc
```

2.162.3 Description

These fixes compute induced charges at the interface between two impermeable media with different dielectric constants. The interfaces need to be discretized into vertices, each representing a boundary element. The vertices are treated as if they were regular atoms or particles. [atom_style dielectric](#) should be used since it defines the additional properties of each interface particle such as interface normal vectors, element areas, and local dielectric mismatch. These fixes also require the use of [pair_style](#) and [kspace_style](#) with the *dielectric* suffix. At every time step, given a configuration of the physical charges in the system (such as atoms and charged particles) these fixes compute and update the charge of the interface particles. The interfaces are allowed to move during the simulation if the appropriate time integrators are also set (for example, with [fix_rigid](#)).

Consider an interface between two media: one with dielectric constant of 78 (water), the other of 4 (silica). The interface is discretized into 2000 boundary elements, each represented by an interface particle. Suppose that each interface particle has a normal unit vector pointing from the silica medium to water. The dielectric difference along the normal vector is then $78 - 4 = 74$, the mean dielectric value is $(78 + 4) / 2 = 41$. Each boundary element also has its area and the local mean curvature, which is used by these fixes for computing a correction term in the local electric field. To model charged interfaces, an interface particle will have a non-zero charge value, coming from its area and surface charge density, and its local dielectric constant set to the mean dielectric value.

For non-interface particles such as atoms and charged particles, the interface normal vectors, element area, and dielectric mismatch are irrelevant and unused. Their local dielectric value is used internally to rescale their given charge when computing the Coulombic interactions. For instance, to simulate a cation carrying a charge of +2 (in simulation charge units) in an implicit solvent with a dielectric constant of 40, the cation's charge should be set to +2 and its local dielectric constant property (defined in the [atom_style dielectric](#)) should be set to 40; there is no need to manually rescale charge. This will produce the proper force for any [pair_style](#) with the dielectric suffix. It is assumed that the particles cannot pass through the interface during the simulation because the value of the local dielectric constant property does not change.

There are some example scripts for using these fixes with LAMMPS in the examples/PACKAGES/dielectric directory. The README file therein contains specific details on the system setup. Note that the example data files show the additional fields (columns) needed for [atom_style dielectric](#) beyond the conventional fields *id*, *mol*, *type*, *q*, *x*, *y*, and *z*.

For fix *polarize/bem/gmres* and fix *polarize/bem/icc* the induced charges of the atoms in the specified group, which are the vertices on the interface, are computed using the equation:

$$\sigma_b(\mathbf{s}) = \frac{1 - \bar{\epsilon}}{\bar{\epsilon}} \sigma_f(\mathbf{s}) - \epsilon_0 \frac{\Delta\epsilon}{\bar{\epsilon}} \mathbf{E}(\mathbf{s}) \cdot \mathbf{n}(\mathbf{s})$$

- σ_b is the induced charge density at the interface vertex \mathbf{s} .
- $\bar{\epsilon}$ is the mean dielectric constant at the interface vertex: $\bar{\epsilon} = (\epsilon_1 + \epsilon_2)/2$.
- $\Delta\epsilon$ is the dielectric constant difference at the interface vertex: $\Delta\epsilon = \epsilon_1 - \epsilon_2$
- σ_f is the free charge density at the interface vertex
- $\mathbf{E}(\mathbf{s})$ is the electrical field at the vertex
- $\mathbf{n}(\mathbf{s})$ is the unit normal vector at the vertex pointing from medium with ϵ_2 to that with ϵ_1

Fix *polarize/bem/gmres* employs the Generalized Minimum Residual (GMRES) as described in ([Barros](#)) to solve σ_b .

Fix *polarize/bem/icc* employs the successive over-relaxation algorithm as described in ([Tyagi](#)) to solve σ_b .

The iterative solvers would terminate either when the maximum relative change in the induced charges in consecutive iterations is below the set tolerance, or when the number of iterations reaches *iter_max* (see below).

Fix *polarize/functional* employs the energy functional variation approach as described in ([Jadhao](#)) to solve σ_b .

The induced charges computed by these fixes are stored in the *q_scaled* field, and can be accessed as in the following example:

```
compute qs all property/atom q_scaled
dump 1 all custom 1000 all.txt id type q x y z c_qs
```

Note that the *q* field is the regular atom charges, which do not change during the simulation. For interface particles, *q_scaled* is the sum of the real charge, divided by the local dielectric constant *epsilon*, and their induced charges. For non-interface particles, *q_scaled* is the real charge, divided by the local dielectric constant *epsilon*.

More details on the implementation of these fixes and their recommended use are described in ([NguyenTD](#)).

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

No information about this fix is written to *binary restart files*.

The *fix_modify* command provides the ability to modify certain settings:

```
itr_max arg
  arg = maximum number of iterations for convergence
dielectrics ediff emean epsilon area charge
  ediff = dielectric difference or NULL
  emean = dielectric mean or NULL
  epsilon = local dielectric value or NULL
  area = element area or NULL
  charge = real interface charge or NULL
kspace arg = yes or no
rand max seed
  max = range of random induced charges to be generated
  seed = random number seed to use when generating random charge
mr arg
  arg = maximum number of q-vectors to use when solving (GMRES only)
omega arg
  arg = relaxation parameter to use when iterating (ICC only)
```

The *itr_max* keyword sets the max number of iterations to be used for solving each step.

The *dielectrics* keyword allows properties of the atoms in group *group-ID* to be modified. Values passed to any of the arguments (*ediff*, *emean*, *epsilon*, *area*, *charge*) will override existing values for all atoms in the group *group-ID*. Passing NULL to any of these arguments will preserve the existing value. Note that setting the properties of the interface this way will change the properties of all atoms associated with the fix (all atoms in *group-ID*), so multiple fix and fix_modify commands would be needed to change the properties of two different interfaces to different values (one fix and fix_modify for each interface group).

The *kspace* keyword turns on long range interactions.

If the arguments of the *rand* keyword are set, then the atoms subject to this fix will be assigned a random initial charge in a uniform distribution from *-max/2* to *max/2*, using random number seed *seed*.

The *mr* keyword only applies to *style = polarize/bem/gmres*. It is the maximum number of q-vectors to use when solving for the surface charge.

The *omega* keyword only applies when using *style = polarize/bem/icc*. It is a relaxation parameter defined in [\(Tyagi\)](#) that should generally be set between 0 and 2.

Note that the local dielectric constant (*epsilon*) can also be set independently using the [set](#) command.

polarize/bem/gmres or *polarize/bem/icc* compute a global 2-element vector which can be accessed by various [output commands](#). The first element is the number of iterations when the solver terminates (of which the upper bound is set by *iter_max*). The second element is the RMS error.

2.162.5 Restrictions

These fixes are part of the DIELECTRIC package. They are only enabled if LAMMPS was built with that package, which requires that also the KSPACE package is installed. See the [Build package](#) page for more info.

Note that the *polarize/bem/gmres* and *polarize/bem/icc* fixes only support [units lj, real, metal, si](#) and *nano* at the moment.

Note that *polarize/functional* does not yet support charged interfaces.

2.162.6 Related commands

pair_coeff, *fix polarize*, *read_data*, *pair_style lj/cut/coul/long/dielectric*, *kspace_style pppm/dielectric*, *compute field/atom*

2.162.7 Default

iter_max = 50

kspace = yes

omega = 0.7 (ICC only)

mr = # atoms in group *group-ID* minus 1 (GMRES only)

No random charge initialization happens by default.

(Barros) Barros, Sinkovits, Luijten, J. Chem. Phys, 140, 064903 (2014)

(Tyagi) Tyagi, Suzen, Sega, Barbosa, Kantorovich, Holm, J Chem Phys, 132, 154112 (2010)

(Jad Hao) Jad Hao, Solis, Olvera de la Cruz, J Chem Phys, 138, 054119 (2013)

(Nguyen TD) Nguyen, Li, Bagchi, Solis, Olvera de la Cruz, Comput Phys Commun 241, 80-19 (2019)

2.163 fix pour command

2.163.1 Syntax

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

- ID, group-ID are documented in [fix](#) command
- pour = style name of this fix command
- N = # of particles to insert
- type = atom type to assign to inserted particles (offset for molecule insertion)
- seed = random # seed (positive integer)
- one or more keyword/value pairs may be appended to args
- keyword = *region* or *diam* or *id* or *vol* or *rate* or *dens* or *vel* or *mol* or *molfrac* or *rigid* or *shake* or *ignore*

region value = region-ID

region-ID = ID of region to use as insertion volume

diam values = dstyle args

dstyle = one or range or poly

one args = D

D = single diameter for inserted particles (distance units)

range args = Dlo Dhi

Dlo,Dhi = range of diameters for inserted particles (distance units)

poly args = Npoly D1 P1 D2 P2 ...

Npoly = # of (D,P) pairs

D1,D2,... = diameter for subset of inserted particles (distance units)

P1,P2,... = percentage of inserted particles with this diameter (0-1)

id values = idflag

idflag = max or next = how to choose IDs for inserted particles and molecules

vol values = fraction Nattempt

fraction = desired volume fraction for filling insertion volume

Nattempts = max # of insertion attempts per particle

rate value = V

V = z velocity (3d) or y velocity (2d) at which

insertion volume moves (velocity units)

dens values = Rholo Rhohi

Rholo,Rhohi = range of densities for inserted particles (mass/volume units)

vel values (3d) = vxlo vxhi vylo vyhi vz

vel values (2d) = vxlo vxhi vy

vxlo,vxhi = range of x velocities for inserted particles (velocity units)

vylo,vyhi = range of y velocities for inserted particles (velocity units)

vz = z velocity (3d) assigned to inserted particles (velocity units)

vy = y velocity (2d) assigned to inserted particles (velocity units)

mol value = template-ID

template-ID = ID of molecule template specified in a separate [molecule](#) command

molfrac values = f1 f2 ... fn

f1 to fn = relative probability of creating each of N molecules in template-ID

rigid value = fix-ID
 fix-ID = ID of fix rigid/small command
 shake value = fix-ID
 fix-ID = ID of fix shake command
 ignore value = none
 skip any line or triangle particles when detecting possible
 overlaps with inserted particles

2.163.2 Examples

```

fix 3 all pour 1000 2 29494 region myblock
fix 2 all pour 10000 1 19985583 region disk vol 0.33 100 rate 1.0 diam range 0.9 1.1
fix 2 all pour 10000 1 19985583 region disk diam poly 2 0.7 0.4 1.5 0.6
fix ins all pour 500 1 4767548 vol 0.8 10 region slab mol object rigid myRigid

```

2.163.3 Description

Insert finite-size particles or molecules into the simulation box every few timesteps within a specified region until N particles or molecules have been inserted. This is typically used to model the pouring of granular particles into a container under the influence of gravity. For the remainder of this doc page, a single inserted atom or molecule is referred to as a “particle”.

If inserted particles are individual atoms, they are assigned the specified atom type. If they are molecules, the type of each atom in the inserted molecule is specified in the file read by the [molecule](#) command, and those values are added to the specified atom type. E.g. if the file specifies atom types 1,2,3, and those are the atom types you want for inserted molecules, then specify *type* = 0. If you specify *type* = 2, the in the inserted molecule will have atom types 3,4,5.

All atoms in the inserted particle are assigned to two groups: the default group “all” and the group specified in the fix pour command (which can also be “all”).

This command must use the *region* keyword to define an insertion volume. The specified region must have been previously defined with a [region](#) command. It must be of type *block* or a z-axis *cylinder* and must be defined with side = *in*. The cylinder style of region can only be used with 3d simulations.

Individual atoms are inserted, unless the *mol* keyword is used. It specifies a *template-ID* previously defined using the [molecule](#) command, which reads a file that defines the molecule. The coordinates, atom types, center-of-mass, moments of inertia, etc, as well as any bond/angle/etc and special neighbor information for the molecule can be specified in the molecule file. See the [molecule](#) command for details. The only settings required to be in this file are the coordinates and types of atoms in the molecule.

If the molecule template contains more than one molecule, the relative probability of depositing each molecule can be specified by the *molfrac* keyword. N relative probabilities, each from 0.0 to 1.0, are specified, where N is the number of molecules in the template. Each time a molecule is inserted, a random number is used to sample from the list of relative probabilities. The N values must sum to 1.0.

If you wish to insert molecules via the *mol* keyword, that will be treated as rigid bodies, use the *rigid* keyword, specifying as its value the ID of a separate [fix rigid/small](#) command which also appears in your input script.

Note

If you wish the new rigid molecules (and other rigid molecules) to be thermostatted correctly via [fix rigid/small/nvt](#) or [fix rigid/small/npt](#), then you need to use the [fix_modify dynamic/dof yes](#) command for the rigid fix. This is to inform that fix that the molecule count will vary dynamically.

If you wish to insert molecules via the *mol* keyword, that will have their bonds or angles constrained via SHAKE, use the *shake* keyword, specifying as its value the ID of a separate *fix shake* command which also appears in your input script.

Each timestep particles are inserted, they are placed randomly inside the insertion volume so as to mimic a stream of poured particles. If they are molecules they are also oriented randomly. Each atom in the particle is tested for overlaps with existing particles, including effects due to periodic boundary conditions if applicable. If an overlap is detected, another random insertion attempt is made; see the *vol* keyword discussion below. The larger the volume of the insertion region, the more particles that can be inserted at any one timestep. Particles are inserted again after enough time has elapsed that the previously inserted particles fall out of the insertion volume under the influence of gravity. Insertions continue every so many timesteps until the desired # of particles has been inserted.

Note

If you are monitoring the temperature of a system where the particle count is changing due to adding particles, you typically should use the *compute_modify dynamic/dof yes* command for the temperature compute you are using.

All other keywords are optional with defaults as shown below.

The *diam* option is only used when inserting atoms and specifies the diameters of inserted particles. There are 3 styles: *one*, *range*, or *poly*. For *one*, all particles will have diameter *D*. For *range*, the diameter of each particle will be chosen randomly and uniformly between the specified *Dlo* and *Dhi* bounds. For *poly*, a series of *Npoly* diameters is specified. For each diameter a percentage value from 0.0 to 1.0 is also specified. The *Npoly* percentages must sum to 1.0. For the example shown above with “diam 2 0.7 0.4 1.5 0.6”, all inserted particles will have a diameter of 0.7 or 1.5. 40% of the particles will be small; 60% will be large.

Note that for molecule insertion, the diameters of individual atoms in the molecule can be specified in the file read by the *molecule* command. If not specified, the diameter of each atom in the molecule has a default diameter of 1.0.

The *id* option has two settings which are used to determine the atom or molecule IDs to assign to inserted particles/molecules. In both cases a check is done of the current system to find the maximum current atom and molecule ID of any existing particle. Newly inserted particles and molecules are assigned IDs that increment those max values. For the *max* setting, which is the default, this check is done at every insertion step, which allows for particles to leave the system, and their IDs to potentially be re-used. For the *next* setting this check is done only once when the fix is specified, which can be more efficient if you are sure particles will not be added in some other way.

The *vol* option specifies what volume fraction of the insertion volume will be filled with particles. For particles with a size specified by the *diam range* keyword, they are assumed to all be of maximum diameter *Dhi* for purposes of computing their contribution to the volume fraction.

The higher the volume fraction value, the more particles are inserted each timestep. Since inserted particles cannot overlap, the maximum volume fraction should be no higher than about 0.6. Each timestep particles are inserted, LAMMPS will make up to a total of *M* tries to insert the new particles without overlaps, where *M* = # of inserted particles * *Nattempt*. If LAMMPS is unsuccessful at completing all insertions, it prints a warning.

The *dens* and *vel* options enable inserted particles to have a range of densities or xy velocities. The specific values for a particular inserted particle will be chosen randomly and uniformly between the specified bounds. Internally, the density value for a particle is converted to a mass, based on the radius (volume) of the particle. The *vz* or *vy* value for option *vel* assigns a z-velocity (3d) or y-velocity (2d) to each inserted particle.

The *rate* option moves the insertion volume in the z direction (3d) or y direction (2d). This enables pouring particles from a successively higher height over time.

The *ignore* option is useful when running a simulation that used line segment (2d) or triangle (3d) particles, typically to define boundaries for spherical granular particles to interact with. See the *atom_style line or tri* command for details. Lines and triangles store their size, and if the size is large it may overlap (in a spherical sense) with the insertion region,

even if the line/triangle is oriented such that there is no actual overlap. This can prevent particles from being inserted. The *ignore* keyword causes the overlap check to skip any line or triangle particles. Obviously you should only use it if there is in fact no overlap of the line or triangle particles with the insertion region.

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

No information about this fix is written to *binary restart files*. This means you must be careful when restarting a pouring simulation, when the restart file was written in the middle of the pouring operation. Specifically, you should use a new fix pour command in the input script for the restarted simulation that continues the operation. You will need to adjust the arguments of the original fix pour command to do this.

Also note that because the state of the random number generator is not saved in restart files, you cannot do “exact” restarts with this fix, where the simulation continues on the same as if no restart had taken place. However, in a statistical sense, a restarted simulation should produce the same behavior if you adjust the fix pour parameters appropriately.

None of the *fix_modify* options are relevant to this fix. This fix computes a global scalar, which can be accessed by various output commands. The scalar is the cumulative number of insertions. 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*.

2.163.5 Restrictions

This fix is part of the GRANULAR package. It is only enabled if LAMMPS was built with that package. See the *Build package* page for more info.

For 3d simulations, a gravity fix in the -z direction must be defined for use in conjunction with this fix. For 2d simulations, gravity must be defined in the -y direction.

The specified insertion region cannot be a “dynamic” region, as defined by the *region* command.

2.163.6 Related commands

fix deposit, fix gravity, region

2.163.7 Default

Insertions are performed for individual particles, i.e. no *mol* setting is defined. If the *mol* keyword is used, the default for *molfrac* is an equal probabilities for all molecules in the template. Additional option defaults are *diam = one 1.0*, *dens = 1.0 1.0*, *vol = 0.25 50*, *rate = 0.0*, *vel = 0.0 0.0 0.0 0.0 0.0* (for 3d), *vel = 0.0 0.0 0.0* (for 2d), and *id = max*.

2.164 fix precession/spin command

2.164.1 Syntax

```
fix ID group precession/spin style args
```

- ID, group are documented in *fix* command
- precession/spin = style name of this fix command

- style = *zeeman* or *anisotropy* or *cubic* or *stt*

zeeman args = H x y z

H = intensity of the magnetic field (in Tesla)

x y z = vector direction of the field

anisotropy args = K x y z

K = intensity of the magnetic anisotropy (in eV)

x y z = vector direction of the anisotropy

cubic args = K1 K2c n1x n1y n1x n2x n2y n2z n3x n3y n3z

K1 and K2c = intensity of the magnetic anisotropy (in eV)

n1x to n3z = three direction vectors of the cubic anisotropy

stt args = J x y z

J = intensity of the spin-transfer torque field

x y z = vector direction of the field

2.164.2 Examples

```
fix 1 all precession/spin zeeman 0.1 0.0 0.0 1.0
fix 1 3 precession/spin anisotropy 0.001 0.0 0.0 1.0
fix 1 iron precession/spin cubic 0.001 0.0005 1.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 1.0
fix 1 all precession/spin zeeman 0.1 0.0 0.0 1.0 anisotropy 0.001 0.0 0.0 1.0
```

2.164.3 Description

This fix applies a precession torque to each magnetic spin in the group.

Style *zeeman* is used for the simulation of the interaction between the magnetic spins in the defined group and an external magnetic field:

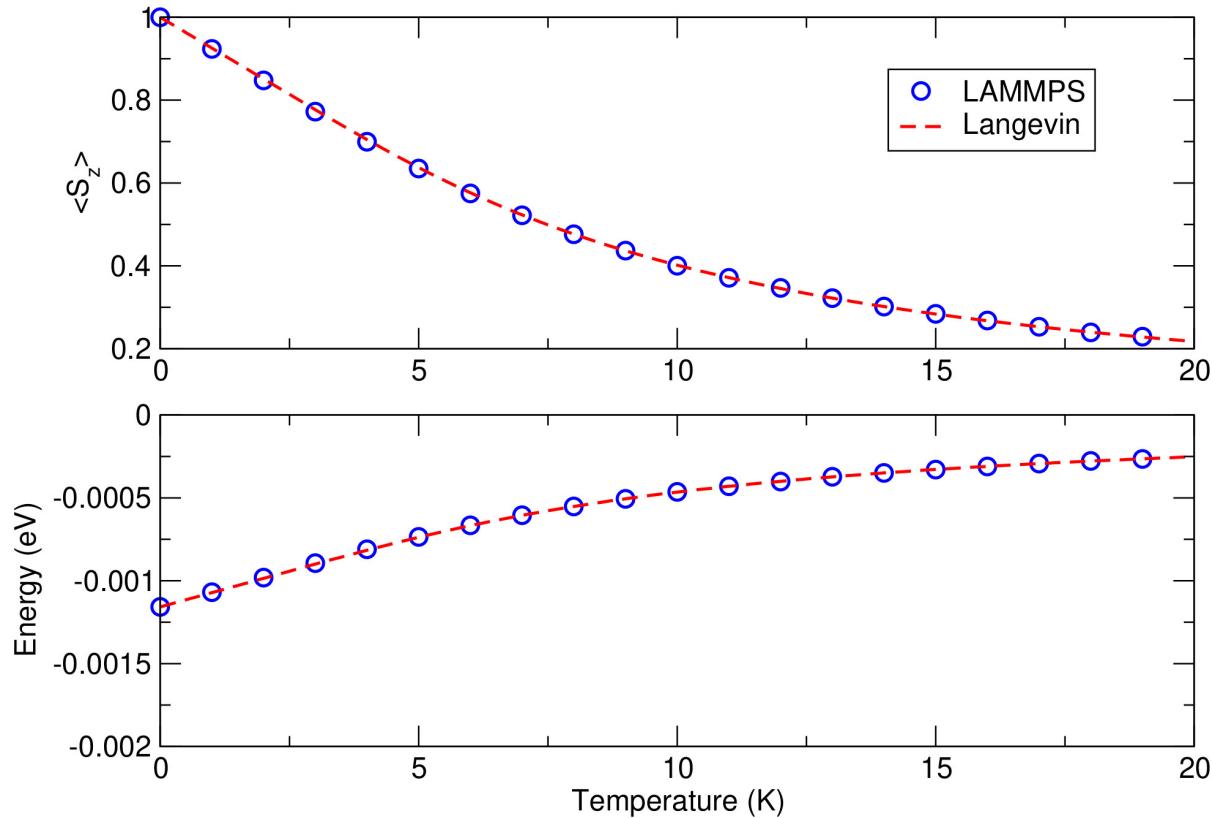
$$H_{Zeeman} = -g \sum_{i=0}^N \mu_i \vec{s}_i \cdot \vec{B}_{ext}$$

with:

- \vec{B}_{ext} the external magnetic field (in T)
- g the Lande factor (hard-coded as g = 2.0)
- \vec{s}_i the unitary vector describing the orientation of spin *i*
- μ_i the atomic moment of spin *i* given as a multiple of the Bohr magneton μ_B (for example, $\mu_i \approx 2.2$ in bulk iron).

The field value in Tesla is multiplied by the gyromagnetic ratio, $g \cdot \mu_B / \hbar$, converting it into a precession frequency in rad.THz (in metal units and with $\mu_B = 5.788 \cdot 10^{-5}$ eV/T).

As a comparison, the figure below displays the simulation of a single spin (of norm $\mu_i = 1.0$) submitted to an external magnetic field of $|B_{ext}| = 10.0$ Tesla (and oriented along the z axis). The upper plot shows the average magnetization along the external magnetic field axis and the lower plot the Zeeman energy, both as a function of temperature. The reference result is provided by the plot of the Langevin function for the same parameters.



The temperature effects are accounted for by connecting the spin i to a thermal bath using a Langevin thermostat (see [fix langevin/spin](#) for the definition of this thermostat).

Style *anisotropy* is used to simulate an easy axis or an easy plane for the magnetic spins in the defined group:

$$H_{\text{aniso}} = - \sum_{i=1}^N K_{\text{an}}(\mathbf{r}_i) (\vec{s}_i \cdot \vec{n}_i)^2$$

with n defining the direction of the anisotropy, and K (in eV) its intensity. If $K > 0$, an easy axis is defined, and if $K < 0$, an easy plane is defined.

Style *cubic* is used to simulate a cubic anisotropy, with three possible easy axis for the magnetic spins in the defined group:

$$H_{\text{cubic}} = - \sum_{i=1}^N K_1 \left[(\vec{s}_i \cdot \vec{n}_1)^2 (\vec{s}_i \cdot \vec{n}_2)^2 + (\vec{s}_i \cdot \vec{n}_2)^2 (\vec{s}_i \cdot \vec{n}_3)^2 + (\vec{s}_i \cdot \vec{n}_3)^2 (\vec{s}_i \cdot \vec{n}_1)^2 \right] + K_2^{(c)} (\vec{s}_i \cdot \vec{n}_1)^2 (\vec{s}_i \cdot \vec{n}_2)^2 (\vec{s}_i \cdot \vec{n}_3)^2$$

with K_1 and $K_2^{(c)}$ (in eV) the intensity coefficients and \vec{n}_1 , \vec{n}_2 and \vec{n}_3 defining the three anisotropic directions defined by the command (from $n1x$ to $n3z$). For $\vec{n}_1 = (100)$, $\vec{n}_2 = (010)$, and $\vec{n}_3 = (001)$, $K_1 < 0$ defines an iron type anisotropy (easy axis along the (001)-type cube edges), and $K_1 > 0$ defines a nickel type anisotropy (easy axis along the (111)-type cube diagonals). $K_2^{(c)} > 0$ also defines easy axis along the (111)-type cube diagonals. See chapter 2 of ([Skomski](#)) for more details on cubic anisotropies.

Style *sst* is used to simulate the interaction between the spins and a spin-transfer torque. See equation (7) of ([Chirac](#)) for more details about the implemented spin-transfer torque term.

In all cases, the choice of (xyz) only imposes the vector directions for the forces. Only the direction of the vector is important; its length is ignored (the entered vectors are normalized).

Those styles can be combined within one single command line.

Note

The norm of all vectors defined with the precession/spin command have to be non-zero. For example, defining “fix 1 all precession/spin zeeman 0.1 0.0 0.0 0.0” would result in an error message. Since those vector components are used to compute the inverse of the field (or anisotropy) vector norm, setting a zero-vector would result in a division by zero.

2.164.4 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 associated with the spin precession torque to the global potential energy of the system as part of *thermodynamic output*. The default setting for this fix is *fix_modify energy no*.

This fix computes a global scalar which can be accessed by various *output commands*. The scalar is the potential energy (in energy units) discussed in the previous paragraph. The scalar value is an “extensive” quantity.

No information about this fix is written to *binary restart files*.

2.164.5 Restrictions

The *precession/spin* style is part of the SPIN package. This style is only enabled if LAMMPS was built with this package, and if the atom_style “spin” was declared. See the *Build package* page for more info.

2.164.6 Related commands

atom_style spin

2.164.7 Default

none

(Skomski) Skomski, R. (2008). Simple models of magnetism. Oxford University Press.

(Chirac) Chirac, Theophile, et al. Ultrafast antiferromagnetic switching in NiO induced by spin transfer torques. Physical Review B 102.13 (2020): 134415.

2.165 fix press/berendsen command

2.165.1 Syntax

```
fix ID group-ID press/berendsen keyword value ...
```

- ID, group-ID are documented in *fix* command
- press/berendsen = style name of this fix command

one or more keyword value pairs may be appended
 keyword = iso or aniso or x or y or z or couple or dilate or modulus
 iso or aniso values = Pstart Pstop Pdamp
 Pstart,Pstop = scalar external pressure at start/end of run (pressure units)
 Pdamp = pressure damping parameter (time units)
 x or y or z values = Pstart Pstop Pdamp
 Pstart,Pstop = external stress tensor component at start/end of run (pressure units)
 Pdamp = stress damping parameter (time units)
 couple = none or xyz or xy or yz or xz
 modulus value = bulk modulus of system (pressure units)
 dilate value = all or partial

2.165.2 Examples

```
fix 1 all press/berendsen iso 0.0 0.0 1000.0
fix 2 all press/berendsen aniso 0.0 0.0 1000.0 dilate partial
```

2.165.3 Description

Reset the pressure of the system by using a Berendsen barostat ([Berendsen](#)), which rescales the system volume and (optionally) the atoms coordinates within the simulation box every timestep.

Regardless of what atoms are in the fix group, a global pressure is computed for all atoms. Similarly, when the size of the simulation box is changed, all atoms are re-scaled to new positions, unless the keyword *dilate* is specified with a value of *partial*, in which case only the atoms in the fix group are re-scaled. The latter can be useful for leaving the coordinates of atoms in a solid substrate unchanged and controlling the pressure of a surrounding fluid.

Note

Unlike the [fix npt](#) or [fix nph](#) commands which perform Nose/Hoover barostatting AND time integration, this fix does NOT perform time integration. It only modifies the box size and atom coordinates to effect barostatting. Thus you must use a separate time integration fix, like [fix nve](#) or [fix nvt](#) to actually update the positions and velocities of atoms. This fix can be used in conjunction with thermostating fixes to control the temperature, such as [fix nvt](#) or [fix langevin](#) or [fix temp/berendsen](#).

See the [Howto barostat](#) page for a discussion of different ways to perform barostatting.

The barostat is specified using one or more of the *iso*, *aniso*, *x*, *y*, *z*, and *couple* keywords. These keywords give you the ability to specify the 3 diagonal components of an external stress tensor, and to couple various of these components together so that the dimensions they represent are varied together during a constant-pressure simulation. Unlike the [fix npt](#) and [fix nph](#) commands, this fix cannot be used with triclinic (non-orthogonal) simulation boxes to control all 6 components of the general pressure tensor.

The target pressures for each of the 3 diagonal components of the stress tensor can be specified independently via the *x*, *y*, *z*, keywords, which correspond to the 3 simulation box dimensions. For each component, the external pressure or tensor component at each timestep is a ramped value during the run from *Pstart* to *Pstop*. If a target pressure is specified for a component, then the corresponding box dimension will change during a simulation. For example, if the *y* keyword is used, the *y*-box length will change. A box dimension will not change if that component is not specified, although you have the option to change that dimension via the [fix deform](#) command.

For all barostat keywords, the *Pdamp* parameter determines the time scale on which pressure is relaxed. For example, a value of 10.0 means to relax the pressure in a timespan of (roughly) 10 time units (tau or fs or ps - see the [units](#) command).

Note

A Berendsen barostat will not work well for arbitrary values of *Pdamp*. If *Pdamp* is too small, the pressure and volume can fluctuate wildly; if it is too large, the pressure will take a very long time to equilibrate. A good choice for many models is a *Pdamp* of around 1000 timesteps. However, note that *Pdamp* is specified in time units, and that timesteps are NOT the same as time units for most [units](#) settings.

Note

The relaxation time is actually also a function of the bulk modulus of the system (inverse of isothermal compressibility). The bulk modulus has units of pressure and is the amount of pressure that would need to be applied (isotropically) to reduce the volume of the system by a factor of 2 (assuming the bulk modulus was a constant, independent of density, which it's not). The bulk modulus can be set via the keyword *modulus*. The *Pdamp* parameter is effectively multiplied by the bulk modulus, so if the pressure is relaxing faster than expected or desired, increasing the bulk modulus has the same effect as increasing *Pdamp*. The converse is also true. LAMMPS does not attempt to guess a correct value of the bulk modulus; it just uses 10.0 as a default value which gives reasonable relaxation for a Lennard-Jones liquid, but will be way off for other materials and way too small for solids. Thus you should experiment to find appropriate values of *Pdamp* and/or the *modulus* when using this fix.

The *couple* keyword allows two or three of the diagonal components of the pressure tensor to be “coupled” together. The value specified with the keyword determines which are coupled. For example, *xz* means the *Pxx* and *Pzz* components of the stress tensor are coupled. *Xyz* means all 3 diagonal components are coupled. Coupling means two things: the instantaneous stress will be computed as an average of the corresponding diagonal components, and the coupled box dimensions will be changed together in lockstep, meaning coupled dimensions will be dilated or contracted by the same percentage every timestep. The *Pstart*, *Pstop*, *Pdamp* parameters for any coupled dimensions must be identical. *Couple xyz* can be used for a 2d simulation; the *z* dimension is simply ignored.

The *iso* and *aniso* keywords are simply shortcuts that are equivalent to specifying several other keywords together.

The keyword *iso* means couple all 3 diagonal components together when pressure is computed (hydrostatic pressure), and dilate/contract the dimensions together. Using “*iso Pstart Pstop Pdamp*” is the same as specifying these 4 keywords:

```
x Pstart Pstop Pdamp
y Pstart Pstop Pdamp
z Pstart Pstop Pdamp
couple xyz
```

The keyword *aniso* means *x*, *y*, and *z* dimensions are controlled independently using the *Pxx*, *Pyy*, and *Pzz* components of the stress tensor as the driving forces, and the specified scalar external pressure. Using “*aniso Pstart Pstop Pdamp*” is the same as specifying these 4 keywords:

```
x Pstart Pstop Pdamp
y Pstart Pstop Pdamp
z Pstart Pstop Pdamp
couple none
```

This fix computes a temperature and pressure each timestep. To do this, the fix creates its own computes of style “temp” and “pressure”, as if these commands had been issued:

```
compute fix-ID_temp group-ID temp
compute fix-ID_press group-ID pressure fix-ID_temp
```

See the [compute temp](#) and [compute pressure](#) commands for details. Note that the IDs of the new computes are the fix-ID + underscore + “temp” or fix_ID + underscore + “press”, and the group for the new computes is the same as the fix group.

Note that these are NOT the computes used by thermodynamic output (see the [thermo_style](#) command) with ID = *thermo_temp* and *thermo_press*. This means you can change the attributes of this fix’s temperature or pressure via the [compute_modify](#) command or print this temperature or pressure during thermodynamic output via the [thermo_style custom](#) command using the appropriate compute-ID. It also means that changing attributes of *thermo_temp* or *thermo_press* will have no effect on this fix.

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

No information about this fix is written to [binary restart files](#).

The [fix_modify temp](#) and [press](#) options are supported by this fix. You can use them to assign a [compute](#) you have defined to this fix which will be used in its temperature and pressure calculations. If you do this, note that the kinetic energy derived from the compute temperature should be consistent with the virial term computed using all atoms for the pressure. LAMMPS will warn you if you choose to compute temperature on a subset of atoms.

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

This fix can ramp its target pressure over multiple runs, using the *start* and *stop* keywords of the [run](#) command. See the [run](#) command for details of how to do this.

This fix is not invoked during [energy minimization](#).

2.165.5 Restrictions

Any dimension being adjusted by this fix must be periodic.

2.165.6 Related commands

[fix nve](#), [fix nph](#), [fix npt](#), [fix temp/berendsen](#), [fix_modify](#)

2.165.7 Default

The keyword defaults are dilate = all, modulus = 10.0 in units of pressure for whatever [units](#) are defined.

(Berendsen) Berendsen, Postma, van Gunsteren, DiNola, Haak, J Chem Phys, 81, 3684 (1984).

2.166 fix press/langevin command

2.166.1 Syntax

```
fix ID group-ID press/langevin keyword value ...
```

- ID, group-ID are documented in [fix](#) command
- press/langevin = 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 xz or yz or couple or dilate or modulus or temp
→ or flip

iso or aniso or tri values = Pstart Pstop Pdamp

Pstart,Pstop = scalar external pressure at start/end of run (pressure units)

Pdamp = pressure damping parameter (time units)

x or y or z or xy or xz or yz values = Pstart Pstop Pdamp

Pstart,Pstop = external stress tensor component at start/end of run (pressure units)

Pdamp = pressure damping parameter

flip value = yes or no = allow or disallow box flips when it becomes highly skewed

couple = none or xyz or xy or yz or xz

friction value = Friction coefficient for the barostat (time units)

temp values = Tstart, Tstop, seed

Tstart, Tstop = target temperature used for the barostat at start/end of run

seed = seed of the random number generator

dilate value = all or partial

2.166.2 Examples

```
fix 1 all press/langevin iso 0.0 0.0 1000.0 temp 300 300 487374
fix 2 all press/langevin aniso 0.0 0.0 1000.0 temp 100 300 238 dilate partial
```

2.166.3 Description

Adjust the pressure of the system by using a Langevin stochastic barostat ([Gronbech](#)), which rescales the system volume and (optionally) the atoms coordinates within the simulation box every timestep.

The Langevin barostat couple each direction L with a pseudo-particle that obeys the Langevin equation such as:

$$\begin{aligned}
 f_P &= \frac{Nk_B T_{target}}{V} + \frac{1}{Vd} \sum_{i=1}^N \vec{r}_i \cdot \vec{f}_i - P_{target} \\
 Q\ddot{L} + \alpha\dot{L} &= f_P + \beta(t) \\
 L^{n+1} &= L^n + bdt \dot{L}^n \frac{bdt^2}{2Q} \\
 \dot{L}^{n+1} &= \alpha\dot{L}^n + \frac{dt}{2Q} (af_P^n + f_P^{n+1}) + \frac{b}{Q}\beta^{n+1} \\
 a &= \frac{1 - \frac{\alpha dt}{2Q}}{1 + \frac{\alpha dt}{2Q}} \\
 b &= \frac{1}{1 + \frac{\alpha dt}{2Q}} \\
 \langle \beta(t)\beta(t') \rangle &= 2\alpha k_B T dt
 \end{aligned}$$

Where dt is the timestep \dot{L} and \ddot{L} the first and second derivatives of the coupled direction with regard to time, α is a friction coefficient, β is a random gaussian variable and Q the effective mass of the coupled pseudoparticle. The two first terms on the right-hand side of the first equation are the virial expression of the canonical pressure. It is to be noted that the temperature used to compute the pressure is not based on the atom velocities but rather on the canonical target temperature directly. This temperature is specified using the *temp* keyword parameter and should be close to the expected target temperature of the system.

Regardless of what atoms are in the fix group, a global pressure is computed for all atoms. Similarly, when the size of the simulation box is changed, all atoms are re-scaled to new positions, unless the keyword *dilate* is specified with a value of *partial*, in which case only the atoms in the fix group are re-scaled. The latter can be useful for leaving the coordinates of atoms in a solid substrate unchanged and controlling the pressure of a surrounding fluid.

Note

Unlike the *fix npt* or *fix nph* commands which perform Nose-Hoover barostatting AND time integration, this fix does NOT perform time integration of the atoms but only of the barostat coupled coordinate. It then only modifies the box size and atom coordinates to effect barostatting. Thus you must use a separate time integration fix, like *fix nve* or *fix nvt* to actually update the positions and velocities of atoms. This fix can be used in conjunction with thermostating fixes to control the temperature, such as *fix nvt* or *fix langevin* or *fix temp/berendsen*.

See the *Howto barostat* page for a discussion of different ways to perform barostatting.

The barostat 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 the 3 diagonal components of an external stress tensor, and to couple various of these components together so that the dimensions they represent are varied together during a constant-pressure simulation.

The target pressures for each of the 6 diagonal components of the stress tensor can be specified independently via the *x*, *y*, *z*, keywords, which correspond to the 3 simulation box dimensions, and the *xy*, *xz* and *yz* keywords which corresponds to the 3 simulation box tilt factors. For each component, the external pressure or tensor component at each timestep is a ramped value during the run from *Pstart* to *Pstop*. If a target pressure is specified for a component, then the corresponding box dimension will change during a simulation. For example, if the *y* keyword is used, the *y*-box length will change. A box dimension will not change if that component is not specified, although you have the option to change that dimension via the *fix deform* command.

The *Pdamp* parameter can be seen in the same way as a Nose-Hoover parameter as it is used to compute the mass of the fictitious particle. Without friction, the barostat can be compared to a single particle Nose-Hoover barostat and should

follow a similar decay in time. The mass of the barostat is linked to *Pdamp* by the relation $Q = (N_{at} + 1) \cdot k_B T_{target} \cdot P_{damp}^2$. Note that *Pdamp* should be expressed in time units.

Note

As for Berendsen barostat, a Langevin barostat will not work well for arbitrary values of *Pdamp*. If *Pdamp* is too small, the pressure and volume can fluctuate wildly; if it is too large, the pressure will take a very long time to equilibrate. A good choice for many models is a *Pdamp* of around 1000 timesteps. However, note that *Pdamp* is specified in time units, and that timesteps are NOT the same as time units for most *units* settings.

The *temp* keyword sets the temperature to use in the equation of motion of the barostat. This value is used to compute the value of the force f_p in the equation of motion. It is important to note that this value is not the instantaneous temperature but a target temperature that ramps from *Tstart* to *Tstop*. Also the required argument *seed* sets the seed for the random number generator used in the generation of the random forces.

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 P_{xx} and P_{zz} components of the stress tensor are coupled. *Xyz* means all 3 diagonal components are coupled. Coupling means two things: the instantaneous stress will be computed as an average of the corresponding diagonal components, and the coupled box dimensions will be changed together in lockstep, meaning coupled dimensions will be dilated or contracted by the same percentage every timestep. The *Pstart*, *Pstop*, *Pdamp* parameters for any coupled dimensions must be identical. *Couple xyz* can be used for a 2d simulation; the *z* dimension is simply ignored.

The *iso*, *aniso* and *tri* keywords are simply shortcuts that are equivalent to specifying several other keywords together. The keyword *iso* means couple all 3 diagonal components together when pressure is computed (hydrostatic pressure), and dilate/contract the dimensions together. Using “*iso Pstart Pstop Pdamp*” is the same as specifying these 4 keywords:

```
x Pstart Pstop Pdamp
y Pstart Pstop Pdamp
z Pstart Pstop Pdamp
couple xyz
```

The keyword *aniso* means *x*, *y*, and *z* dimensions are controlled independently using the P_{xx} , P_{yy} , and P_{zz} components of the stress tensor as the driving forces, and the specified scalar external pressure. Using “*aniso Pstart Pstop Pdamp*” is the same as specifying these 4 keywords:

```
x Pstart Pstop Pdamp
y Pstart Pstop Pdamp
z Pstart Pstop Pdamp
couple none
```

The keyword *tri* is the same as *aniso* but also adds the control on the shear pressure coupled with the tilt factors.

```
x Pstart Pstop Pdamp
y Pstart Pstop Pdamp
z Pstart Pstop Pdamp
xy Pstart Pstop Pdamp
xz Pstart Pstop Pdamp
yz Pstart Pstop Pdamp
couple none
```

The *flip* keyword allows the tilt factors for a triclinic box to exceed half the distance of the parallel box length, as discussed below. 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 applied stress induces large deformations (e.g. in a liquid), this means the box shape can tilt dramatically and LAMMPS will run less efficiently, due to the large volume of communication needed to acquire ghost atoms around a processor's irregular-shaped subdomain. For extreme values of tilt, LAMMPS may also lose atoms and generate an error.

The *friction* keyword sets the friction parameter α in the equations of motion of the barostat. For each barostat direction, the value of α depends on both *Pdamp* and *friction*. The value given as a parameter is the Langevin characteristic time $\tau_L = \frac{Q}{\alpha}$ in time units. The langevin time can be understood as a decorrelation time for the pressure. A long Langevin time value will make the barostat act as an underdamped oscillator while a short value will make it act as an overdamped oscillator. The ideal configuration would be to find the critical parameter of the barostat. Empirically this is observed to occur for $\tau_L \approx P_{damp}$. For this reason, if the *friction* keyword is not used, the default value *Pdamp* is used for each barostat direction.

This fix computes pressure each timestep. To do this, the fix creates its own computes of style “pressure”, as if this command had been issued:

```
compute fix-ID_press group-ID pressure NULL virial
```

The kinetic contribution to the pressure is taken as the ensemble value $\frac{Nk_bT}{V}$ and computed by the fix itself.

See the [compute pressure](#) command for details. Note that the IDs of the new compute is the fix-ID + underscore + “press” and the group for the new computes is the same as the fix group.

Note that this is NOT the compute used by thermodynamic output (see the [thermo_style](#) command) with ID = *thermo_press*. This means you can change the attributes of this fix’s 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.

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

No information about this fix is written to [binary restart files](#).

The [fix_modify press](#) option is supported by this fix. You can use it to assign a [compute](#) you have defined to this fix which will be used in its pressure calculations.

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

This fix can ramp its target pressure and temperature over multiple runs, using the *start* and *stop* keywords of the [run](#) command. See the [run](#) command for details of how to do this. It is recommended that the ramped temperature is the same as the effective temperature of the thermostatted system. That is, if the system’s temperature is ramped by other commands, it is recommended to do the same with this pressure control.

This fix is not invoked during [energy minimization](#).

2.166.5 Restrictions

Any dimension being adjusted by this fix must be periodic.

2.166.6 Related commands

fix press/berendsen, fix nve, fix nph, fix npt, fix langevin, fix_modify

2.166.7 Default

The keyword defaults are *dilate* = all, *flip* = yes, and *friction* = *Pdamp*.

(Gronbech) Gronbech-Jensen, Farago, J Chem Phys, 141, 194108 (2014).

2.167 fix print command

2.167.1 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; N can be a variable (see below)
 - 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

2.167.2 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
```

2.167.3 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 single or double quotes if it is more than one word. If it contains variables it must be enclosed in double quotes to ensure they are not evaluated when the input script line is read, but will instead be evaluated each time the string is printed.

Added in version 15Jun2023: support for vector style variables

See the [variable](#) command for a description of *equal* and *vector* style variables which are typically the most useful ones to use with the print command. Equal- and vector-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](#). Vector-style variables are printed in a bracketed, comma-separated format, e.g. [1,2,3,4] or [12.5,2,4.6,10.1].

Note

As discussed on the [Commands parse](#) doc page, the text string can use “immediate” variables, specified as \$(formula) with parenthesis, where the numeric formula has the same syntax as equal-style variables described on the [variable](#) doc page. This is a convenient way to evaluate a formula immediately without using the variable command to define a named variable and then use that variable in the text string. The formula can include a trailing colon and format string which determines the precision with which the numeric value is output. This is also explained on the [Commands parse](#) doc page.

Instead of a numeric value, N can be specified 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 the string 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. For example, the following commands will print output at timesteps 10,20,30,100,200,300,1000,2000,etc:

```
variable      s equal logfreq(10,3,10)
fix extra all print v_s "Coords of marker atom = $x $y $z"
```

The specified group-ID is ignored by this fix.

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.

2.167.4 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*.

2.167.5 Restrictions

none

2.167.6 Related commands

variable, *print*

2.167.7 Default

The option defaults are no file output, screen = yes, and title string as described above.

2.168 fix propel/self command

2.168.1 Syntax

```
fix ID group-ID propel/self mode magnitude keyword values
```

- ID, group-ID are documented in *fix* command
- propel/self = style name of this fix command
- mode = *dipole* or *velocity* or *quat*
- magnitude = magnitude of self-propulsion force
- zero or one keyword/value pairs may be appended
- keyword = *qvector*

qvector value = direction of force in ellipsoid frame
sx, *sy*, *sz* = components of *qvector*

2.168.2 Examples

```
fix active all propel/self dipole 40.0
fix active all propel/self velocity 10.0
fix active all propel/self quat 15.7 qvector 1.0 0.0 0.0
```

2.168.3 Description

Add a force to each atom in the group due to a self-propulsion force. The force is given by

$$F_i = f_P e_i$$

where i is the particle the force is being applied to, f_P is the magnitude of the force, and e_i is the vector direction of the force. The specification of e_i is based on which of the three keywords (*dipole* or *velocity* or *quat*) one selects.

For mode *dipole*, e_i is just equal to the dipole vectors of the atoms in the group. Therefore, if the dipoles are not unit vectors, the e_i will not be unit vectors.

Note

If another command changes the magnitude of the dipole, this force will change accordingly (since $|e_i|$ will change, which is physically equivalent to re-scaling f_P while keeping $|e_i|$ constant), and no warning will be provided by LAMMPS. This is almost never what you want, so ensure you are not changing dipole magnitudes with another LAMMPS fix or pair style. Furthermore, self-propulsion forces (almost) always set e_i to be a unit vector for all times, so it's best to set all the dipole magnitudes to 1.0 unless you have a good reason not to (see the [set](#) command on how to do this).

For mode *velocity*, e_i points in the direction of the current velocity (a unit-vector). This can be interpreted as a velocity-dependent friction, as proposed by e.g. ([Erdmann](#)).

For mode *quat*, e_i points in the direction of a unit vector, oriented in the coordinate frame of the ellipsoidal particles, which defaults to point along the x-direction. This default behavior can be changed by via the *quatvec* keyword.

The optional *quatvec* keyword specifies the direction of self-propulsion via a unit vector (sx,sy,sz). The arguments *sx*, *sy*, and *sz*, are defined within the coordinate frame of the atom's ellipsoid. For instance, for an ellipsoid with long axis along its x-direction, if one wanted the self-propulsion force to also be along this axis, set *sx* equal to 1 and *sy*, *sz* both equal to zero. This keyword may only be specified for mode *quat*.

Note

In using keyword *quatvec*, the three arguments *sx*, *sy*, and *sz* will be automatically normalized to components of a unit vector internally to avoid users having to explicitly do so themselves. Therefore, in mode *quat*, the vectors e_i will always be of unit length.

Along with adding a force contribution, this fix can also contribute to the virial (pressure) of the system, defined as $f_P \sum_i \langle e_i \cdot r_i \rangle / (dV)$, where r_i is the *unwrapped* coordinate of particle i in the case of periodic boundary conditions. See ([Winkler](#)) for a discussion of this active pressure contribution.

For modes *dipole* and *quat*, this fix is by default included in pressure computations.

For mode *velocity*, this fix is by default not included in pressure computations.

Note

In contrast to equilibrium systems, pressure of active systems in general depends on the geometry of the container. The active pressure contribution as calculated in this fix is only valid for certain boundary conditions (spherical walls, rectangular walls, or periodic boundary conditions). For other geometries, the pressure must be measured via explicit calculation of the force per unit area on a wall, and so one must not calculate it using this fix. (Use *fix_modify* as described below to turn off the virial contribution of this fix). Again, see ([Winkler](#)) for discussion of why this is the case.

Furthermore, when dealing with active systems, the temperature is no longer well defined. Therefore, one should ensure that the *virial* flag is used in the [compute pressure](#) command (turning off temperature contributions).

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

No information about this fix is written to [binary restart files](#).

The [fix_modify virial](#) option is supported by this fix to add the contribution due to the added forces on atoms to the system's virial as part of [thermodynamic output](#). The default is *virial yes* for keywords *dipole* and *quat*. The default is *virial no* for keyword *velocity*.

No parameter of this fix can be used with the *start/stop* keywords of the [run](#) command.

2.168.5 Restrictions

With keyword *dipole*, this fix only works when the DIPOLE package is enabled. See the [Build package](#) page for more info.

This fix is part of the BROWNIAN package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) doc page for more info.

2.168.6 Related commands

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

2.168.7 Default

none

(Erdmann) U. Erdmann , W. Ebeling, L. Schimansky-Geier, and F. Schweitzer, Eur. Phys. J. B 15, 105-113, 2000.

(Winkler) Winkler, Wysocki, and Gompper, Soft Matter, 11, 6680 (2015).

2.169 fix property/atom command

Accelerator Variants: *property/atom/kk*

2.169.1 Syntax

```
fix ID group-ID property/atom name1 name2 ... keyword value ...
```

- ID, group-ID are documented in [fix](#) command
- property/atom = style name of this fix command
- name1,name2,... = *mol* or *q* or *rmass* or *i_name* or *d_name* or *i2_name* or *d2_name*

mol = molecule IDs
 q = charge
 rmass = per-atom mass
 temperature = internal temperature of atom
 heatflow = internal heat flow of atom
 i_name = new integer vector referenced by name
 d_name = new floating-point vector referenced by name
 i2_name = new integer array referenced by name
 i2_name arg = N = number of columns in the array
 d2_name = new floating-point array referenced by name
 d2_name arg = N = number of columns in the array

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

ghost value = no or yes for whether ghost atom info is communicated

2.169.2 Examples

```
fix 1 all property/atom mol
fix 1 all property/atom i_myflag1 i_myflag2
fix 1 all property/atom d2_sxyz 3 ghost yes
```

2.169.3 Description

Create one or more additional per-atom vectors or arrays to store information about atoms and to use during a simulation. The specified *group-ID* is ignored by this fix.

The atom style used for a simulation defines a set of per-atom properties, as explained on the [atom_style](#) and [read_data](#) doc pages. The latter command defines these properties for each atom in the system when a data file is read. This fix augments the set of per-atom properties with new custom ones. This can be useful in several scenarios.

If the atom style does not define molecule IDs, per-atom charge, per-atom mass, internal temperature, or internal heat flow, they can be added using the *mol*, *q*, *rmass*, *temperature*, or *heatflow* keywords. This could be useful to define “molecules” to use as rigid bodies with the [fix rigid](#) command, or to carry around an extra flag with atoms (stored as a molecule ID) that can be used by various commands like [compute chunk/atom](#) to group atoms without having to use the group command (which is limited to a total of 32 groups including *all*). For finite-size particles, an internal temperature and heat flow can be used to model heat conduction as in the [GRANULAR package](#).

Another application is to use the *rmass* flag in order to have per-atom masses instead of per-type masses. This could be used to study isotope effects with partial isotope substitution. *See below* for an example of simulating a mixture of light and heavy water with the TIP4P water potential.

An alternative to using `fix property/atom` for these examples is to use an atom style that does define molecule IDs or charge or per-atom mass (indirectly via diameter and density) or to use a hybrid atom style that combines two or more atom styles to provide the union of all their atom properties. However, this has two practical drawbacks: first, it typically necessitates changing the format of the Atoms section in the data file and second, it may define additional properties that are not needed such as bond lists, which incurs some overhead when there are no bonds.

In the future, we may add additional existing per-atom properties to fix `property/atom`, similar to *mol*, *q*, *rmass*, *temperature*, or *heatflow* which “turn-on” specific properties defined by some atom styles, so they can be easily used by atom styles that do not define them.

More generally, the *i_name* and *d_name* options allow one or more new custom per-atom vectors to be defined. Likewise the *i2_name* and *d2_name* options allow one or more custom per-atom arrays to be defined. The *i2_name* and *d2_name*

options take an argument N which specifies the number of columns in the per-atom array, i.e. the number of attributes associated with each atom. $N \geq 1$ is required.

Each name must be unique and can use alphanumeric or underscore characters. These vectors and arrays can store whatever values you decide are useful in your simulation. As explained below there are several ways to initialize, access, and output these values, via input script commands, data files, and in new code you add to LAMMPS.

This is effectively a simple way to add per-atom properties to a model without needing to write code for a new [atom style](#) that defines the properties. Note however that implementing a new atom style allows new atom properties to be more tightly and seamlessly integrated with the rest of the code.

The new atom properties encode values that migrate with atoms to new processors and are written to restart files. If you want the new properties to also be defined for ghost atoms, then use the *ghost* keyword with a value of *yes*. This will invoke extra communication when ghost atoms are created (at every re-neighboring) to ensure the new properties are also defined for the ghost atoms.

Properties on ghost atoms

If you use the *mol*, *q* or *rmass* names, you most likely want to set *ghost* yes, since these properties are stored with ghost atoms if you use an [atom style](#) that defines them. Many LAMMPS operations that use molecule IDs or charge, such as neighbor lists and pair styles, will expect ghost atoms to have these values. LAMMPS will issue a warning if you define those vectors but do not set *ghost* yes.

Limitations on ghost atom properties

The specified properties for ghost atoms are not updated every timestep, but only once every few steps when neighbor lists are re-built. Thus the *ghost* keyword is suitable for static properties, like molecule IDs, but not for dynamic properties that change every step. For the latter, the code you add to LAMMPS to change the properties will also need to communicate their new values to/from ghost atoms, an operation that can be invoked from within a [pair style](#) or [fix](#) or [compute](#) that you write.

This fix is one of a small number that can be defined in an input script before the simulation box is created or atoms are defined. This is so it can be used with the [read_data](#) command as described next.

Per-atom properties that are defined by the [atom style](#) are initialized when atoms are created, e.g. by the [read_data](#) or [create_atoms](#) commands. The per-atom properties defined by this fix are not. So you need to initialize them explicitly. One way to do this is [read_data](#) command, using its *fix* keyword and passing it the fix-ID of this fix.

Thus these commands:

```
fix prop all property/atom mol d_flag
read_data data.txt fix prop NULL Molecules
```

would allow a data file to have a section like this:

Molecules

```
1 4 1.5
2 4 3.0
3 10 1.0
4 10 1.0
5 10 1.0
```

(continues on next page)

(continued from previous page)

```
...
N 763 4.5
```

where N is the number of atoms, the first field on each line is the atom-ID, the next two are a molecule-ID and a floating point value that will be stored in a new property called “flag”. If a per-atom array was specified in the fix property/atom command then the N values for that array must be specified consecutively for that property on each line. Note that the order of values on each line corresponds to the order of custom names in the fix property/atom command.

Note that the lines of per-atom properties can be listed in any order. Also note that all the per-atom properties specified by the fix ID (prop in this case) must be included on each line in the specified data file section (Molecules in this case).

Another way of initializing the new properties is via the [set](#) command. For example, if you wanted molecules defined for every set of 10 atoms, based on their atom-IDs, these commands could be used:

```
fix prop all property/atom mol
variable cluster atom ((id-1)/10)+1
set atom * mol v_cluster
```

The *atom-style variable* will create values for atoms with IDs 31,32,33,...40 that are 4.0,4.1,4.2,...,4.9. When the [set](#) commands assigns them to the molecule ID for each atom, they will be truncated to an integer value, so atoms 31-40 will all be assigned a molecule ID of 4.

Note that *atomfile-style variables* can also be used in place of atom-style variables, which means in this case that the molecule IDs could be read-in from a separate file and assigned by the [set](#) command. This allows you to initialize new per-atom properties in a completely general fashion.

For new atom properties specified as *i_name*, *d_name*, *i2_name*, or *d2_name*, the [dump custom](#) and [compute property/atom](#) commands can access their values. This means that the values can be used accessed by fixes like [fix ave/atom](#), accessed by other computes like [compute reduce](#), or used in *atom-style variables*.

For example, these commands will output both the instantaneous and time-averaged values of two new properties to a custom dump file:

```
fix myprops all property/atom i_flag1 d_flag2
compute 1 all property/atom i_flag1 d_flag2
fix 1 all ave/atom 10 10 100 c_1[1] c_1[2]
dump 1 all custom 100 tmp.dump id x y z i_flag1 d_flag2 f_1[1] f_1[2]
```

If you wish to add new *pair styles*, *fixes*, or *computes* that use the per-atom properties defined by this fix, see the [Modify atom](#) doc page which has details on how the custom properties of this fix can be accessed from added classes.

Here is an example of using per-atom masses with TIP4P water to study isotope effects. When setting up simulations with the *TIP4P pair styles* for water, you have to provide exactly one atom type each to identify the water oxygen and hydrogen atoms. Since the atom mass is normally tied to the atom type, this makes it impossible to study multiple isotopes in the same simulation. With *fix property/atom rmass* however, the per-type masses are replaced by per-atom masses. Assuming you have a working input deck for regular TIP4P water, where water oxygen is atom type 1 and water hydrogen is atom type 2, the following lines of input script convert this to using per-atom masses:

```
fix Isotopes all property/atom rmass ghost yes
set type 1 mass 15.9994
set type 2 mass 1.008
```

When writing out the system data with the `write_data` command, there will be a new section named with the fix-ID (i.e. *Isotopes* in this case). Alternatively, you can take an existing data file and just add this *Isotopes* section with one line per atom containing atom-ID and mass. Either way, the extended data file can be read back with:

```
fix Isotopes all property/atom rmass ghost yes
read_data tip4p-isotopes.data fix Isotopes NULL Isotopes
```

Please note that the first *Isotopes* refers to the fix-ID and the second to the name of the section. The following input script code will now change the first 100 water molecules in this example to heavy water:

```
group hwat id 2:300:3
group hwat id 3:300:3
set group hwat mass 2.0141018
```

Styles with a *gpu*, *intel*, *kk*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed on the [Accelerator packages](#) page. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the GPU, INTEL, KOKKOS, OPENMP, and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the [Build package](#) page for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the `-suffix command-line switch` when you invoke LAMMPS, or you can use the `suffix` command in your input script.

See the [Accelerator packages](#) page for more instructions on how to use the accelerated styles effectively.

2.169.4 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.

Warning

When reading data from a restart file, this fix command has to be specified **after** the `read_restart` command and **exactly** the same was in the input script that created the restart file. LAMMPS will only check whether a fix is of the same style and has the same fix ID and in case of a match will then try to initialize the fix with the data stored in the binary restart file. If the names and associated data types in the new fix property/atom command do not match the old one exactly, data can be corrupted or LAMMPS may crash. If the fix is specified **before** the `read_restart` command its data will not be restored.

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`.

2.169.5 Restrictions

none

2.169.6 Related commands

read_data, set, compute property/atom

2.169.7 Default

The default keyword value is ghost = no.

2.170 fix python/invoke command

2.170.1 Syntax

```
fix ID group-ID python/invoke N callback function_name
```

- ID, group-ID are ignored by this fix
- python/invoke = style name of this fix command
- N = execute every N steps
- callback = *post_force* or *end_of_step*
 - post_force = callback after force computations on atoms every N time steps
 - end_of_step = callback after every N time steps

2.170.2 Examples

```
python post_force_callback here """
from lammps import lammps

def post_force_callback(lammps_ptr, vflag):
    lmp = lammps(ptr=lammps_ptr)
    # access LAMMPS state using Python interface
"""

python end_of_step_callback here """
def end_of_step_callback(lammps_ptr):
    lmp = lammps(ptr=lammps_ptr)
    # access LAMMPS state using Python interface
"""

fix pf all python/invoke 50 post_force post_force_callback
fix eos all python/invoke 50 end_of_step end_of_step_callback
```

2.170.3 Description

This fix allows you to call a Python function during a simulation run. The callback is either executed after forces have been applied to atoms or at the end of every N time steps.

Callback functions must be declared in the global scope of the active Python interpreter. This can either be done by defining it inline using the `python` command or by importing functions from other Python modules. If LAMMPS is driven using the library interface from Python, functions defined in the driving Python interpreter can also be executed.

Each callback is given a pointer object as first argument. This can be used to initialize an instance of the `lammps` Python interface, which gives access to the LAMMPS state from Python.

Warning

While you can access the state of LAMMPS via library functions from these callbacks, trying to execute input script commands will in the best case not work or in the worst case result in undefined behavior.

2.170.4 Restrictions

This fix is part of the PYTHON package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

Building LAMMPS with the PYTHON package will link LAMMPS with the Python library on your system. Settings to enable this are in the `lib/python/Makefile.lammps` file. See the `lib/python/README` file for information on those settings.

2.170.5 Related commands

python command

2.171 fix python/move command

2.171.1 Syntax

```
fix python/move pymodule.CLASS
```

`pymodule.CLASS` = use class **CLASS** in module/file **pymodule** to compute how to move atoms

2.171.2 Examples

```
fix 1 all python/move py_nve.NVE
fix 1 all python/move py_nve.NVE_OPT
```

2.171.3 Description

The *python/move* fix style provides a way to define ways how particles are moved during an MD run from python script code, that is loaded from a file into LAMMPS and executed at the various steps where other fixes can be executed. This python script must contain specific python class definitions.

This allows to implement complex position updates and also modified time integration methods. Due to python being an interpreted language, however, the performance of this fix can be moderately to significantly slower than the corresponding C++ code. For specific cases, this performance penalty can be limited through effective use of NumPy.

The python module file has to start with the following code:

```
from __future__ import print_function
import lammps
import ctypes
import traceback
import numpy as np
#
class LAMMPSFix(object):
    def __init__(self, ptr, group_name="all"):
        self.lmp = lammps.lammps(ptr=ptr)
        self.group_name = group_name
#
class LAMMPSFixMove(LAMMPSFix):
    def __init__(self, ptr, group_name="all"):
        super(LAMMPSFixMove, self).__init__(ptr, group_name)
    #
    def init(self):
        pass
    #
    def initial_integrate(self, vflag):
        pass
    #
    def final_integrate(self):
        pass
    #
    def initial_integrate_respa(self, vflag, ilevel, iloop):
        pass
    #
    def final_integrate_respa(self, ilevel, iloop):
        pass
    #
    def reset_dt(self):
        pass
```

Any classes implementing new atom motion functionality have to be derived from the **LAMMPSFixMove** class, overriding the available methods as needed.

Examples for how to do this are in the *examples/python* folder.

2.171.4 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](#).

2.171.5 Restrictions

This pair style is part of the PYTHON package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

2.171.6 Related commands

[fix nve](#), [fix python/invoke](#)

2.171.7 Default

none

2.172 fix qbmsst command

2.172.1 Syntax

`fix ID group-ID qbmsst dir shockvel keyword value ...`

- ID, group-ID are documented in [fix](#) command
 - qbmsst = style name of this fix
 - dir = *x* or *y* or *z*
 - shockvel = shock velocity (strictly positive, velocity units)
 - zero or more keyword/value pairs may be appended
 - keyword = *q* or *mu* or *p0* or *v0* or *e0* or *tscale* or *damp* or *seed* or *f_max* or *N_f* or *eta* or *beta* or *T_init*
- q* value = cell mass-like parameter (mass²/distance⁴ units)
mu value = artificial viscosity (mass/distance/time units)
p0 value = initial pressure in the shock equations (pressure units)
v0 value = initial simulation cell volume in the shock equations (distance³ units)
e0 value = initial total energy (energy units)
tscale value = reduction in initial temperature (unitless fraction between 0.0 and 1.0)
damp value = damping parameter (time units) inverse of friction gamma
seed value = random number seed (positive integer)
f_max value = upper cutoff frequency of the vibration spectrum (1/time units)
N_f value = number of frequency bins (positive integer)
eta value = coupling constant between the shock system and the quantum thermal bath (positive unitless)
beta value = the quantum temperature is updated every beta time steps (positive integer)
T_init value = quantum temperature for the initial state (temperature units)

2.172.2 Examples

```
# (liquid methane modeled with the REAX force field, real units)
fix 1 all qbmsst z 0.122 q 25 mu 0.9 tscale 0.01 damp 200 seed 35082 f_max 0.3 N_f 100 eta 1 beta 400
  ↪ T_init 110
# (quartz modeled with the BKS force field, metal units)
fix 2 all qbmsst z 72 q 40 tscale 0.05 damp 1 seed 47508 f_max 120.0 N_f 100 eta 1.0 beta 500 T_init 300
```

Two example input scripts are given, including shocked α -quartz and shocked liquid methane. The input script first equilibrates an initial state with the quantum thermal bath at the target temperature and then applies *fix qbmsst* to simulate shock compression with quantum nuclear correction. The following two figures plot relevant quantities for shocked α -quartz.

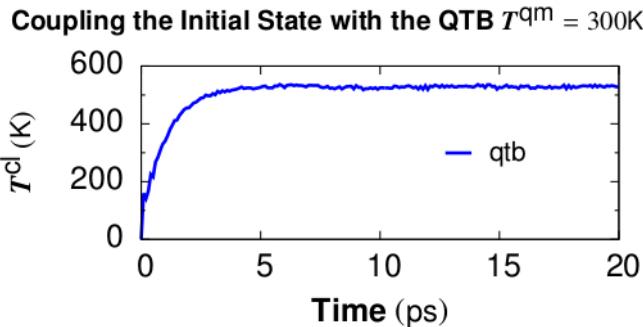


Figure 1. Classical temperature $T_{cl} = \sum \frac{m_i v_i^2}{3Nk_B}$ vs. time for coupling the α -quartz initial state with the quantum thermal bath at target quantum temperature $T^{qm} = 300K$. The NpH ensemble is used for time integration while QTB provides the colored random force. T^{cl} converges at the timescale of *damp* which is set to be 1 ps.

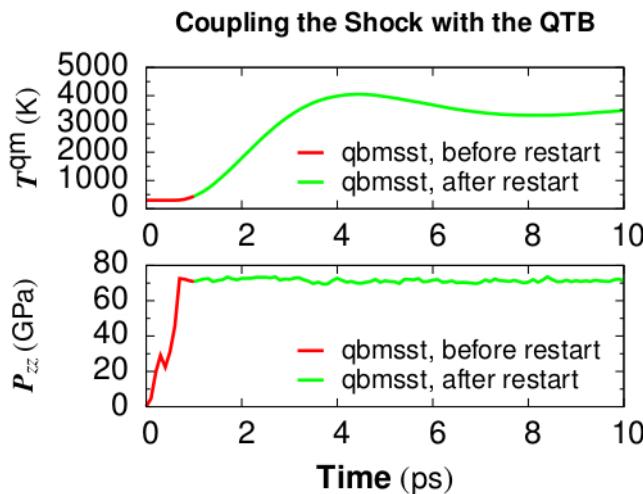


Figure 2. Quantum temperature and pressure vs. time for simulating shocked α -quartz with *fix qbmsst*. The shock propagates along the z direction. Restart of the *fix qbmsst* command is demonstrated in the example input script. Thermodynamic quantities stay continuous before and after the restart.

2.172.3 Description

This command performs the Quantum-Bath coupled Multi-Scale Shock Technique (QBMSST) integration. See ([Qi](#)) for a detailed description of this method. QBMSST provides description of the thermodynamics and kinetics of shock processes while incorporating quantum nuclear effects. The *shockvel* setting determines the steady shock velocity that will be simulated along direction *dir*.

Quantum nuclear effects ([fix qtb](#)) can be crucial especially when the temperature of the initial state is below the classical limit or there is a great change in the zero point energies between the initial and final states. Theoretical post processing quantum corrections of shock compressed water and methane have been reported as much as 30% of the temperatures ([Goldman](#)). A self-consistent method that couples the shock to a quantum thermal bath described by a colored noise Langevin thermostat has been developed by Qi et al ([Qi](#)) and applied to shocked methane. The onset of chemistry is reported to be at a pressure on the shock Hugoniot that is 40% lower than observed with classical molecular dynamics.

It is highly recommended that the system be already in an equilibrium state with a quantum thermal bath at temperature of *T_init*. The fix command [fix qtb](#) at constant temperature *T_init* could be used before applying this command to introduce self-consistent quantum nuclear effects into the initial state.

The parameters *q*, *mu*, *e0*, *p0*, *v0* and *tscale* are described in the command [fix msst](#). The values of *e0*, *p0*, or *v0* will be calculated on the first step if not specified. The parameter of *damp*, *f_max*, and *N_f* are described in the command [fix qtb](#).

The [fix qbmsst](#) command couples the shock system to a quantum thermal bath with a rate that is proportional to the change of the total energy of the shock system, $E^{tot} - E_0^{tot}$. Here E^{tot} consists of both the system energy and a thermal term, see ([Qi](#)), and $E_0^{tot} = e0$ is the initial total energy.

The *eta* (η) parameter is a unitless coupling constant between the shock system and the quantum thermal bath. A small η value cannot adjust the quantum temperature fast enough during the temperature ramping period of shock compression while large η leads to big temperature oscillation. A value of η between 0.3 and 1 is usually appropriate for simulating most systems under shock compression. We observe that different values of η lead to almost the same final thermodynamic state behind the shock, as expected.

The quantum temperature is updated every *beta* (β) steps with an integration time interval β times longer than the simulation time step. In that case, E^{tot} is taken as its average over the past β steps. The temperature of the quantum thermal bath T^{qm} changes dynamically according to the following equation where Δ_t is the MD time step and γ is the friction constant which is equal to the inverse of the *damp* parameter.

$$\frac{dT^{qm}}{dt} = \gamma\eta \sum_{l=1}^{\beta} \frac{E^{tot}(t - l\Delta_t) - E_0^{tot}}{3\beta N k_B}$$

The parameter *T_init* is the initial temperature of the quantum thermal bath and the system before shock loading.

For all pressure styles, the simulation box stays orthorhombic in shape. Parrinello-Rahman boundary conditions (tilted box) are supported by LAMMPS, but are not implemented for QBMSST.

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

Because the state of the random number generator is not written to [binary restart files](#), this fix cannot be restarted “exactly” in an uninterrupted fashion. However, in a statistical sense, a restarted simulation should produce similar behaviors of the system as if it is not interrupted. To achieve such a restart, one should write explicitly the same value for *q*, *mu*, *damp*, *f_max*, *N_f*, *eta*, and *beta* and set *tscale* = 0 if the system is compressed during the first run.

The cumulative energy change in the system imposed by this fix is included in the [thermodynamic output](#) keywords *ecouple* and *econserve*. See the [thermo_style](#) doc page for details.

This fix computes a global scalar which can be accessed by various [output commands](#). The scalar is the same cumulative energy change due to this fix described in the previous paragraph. The scalar value calculated by this fix is “extensive”.

The progress of the QBMSST can be monitored by printing the global scalar and global vector quantities computed by the fix.

As mentioned above, the scalar is the cumulative energy change due to the fix. By monitoring the thermodynamic *econserve* output, this can be used to test if the MD timestep is sufficiently small for accurate integration of the dynamic equations.

The global vector contains five values in the following order. The vector values output by this fix are “intensive”.

`[dhugoniot, drayleigh, lagrangian_speed, lagrangian_position, quantum_temperature]`

1. *dhugoniot* is the departure from the Hugoniot (temperature units).
2. *drayleigh* is the departure from the Rayleigh line (pressure units).
3. *lagrangian_speed* is the laboratory-frame Lagrangian speed (particle velocity) of the computational cell (velocity units).
4. *lagrangian_position* is the computational cell position in the reference frame moving at the shock speed. This is the distance of the computational cell behind the shock front.
5. *quantum_temperature* is the temperature of the quantum thermal bath T^{qm} .

To print these quantities to the log file with descriptive column headers, the following LAMMPS commands are suggested.

```
fix      fix_id all msst z
variable  dhug   equal f_fix_id[1]
variable  dray   equal f_fix_id[2]
variable  lgr_vel equal f_fix_id[3]
variable  lgr_pos equal f_fix_id[4]
variable  T_qm   equal f_fix_id[5]
thermo_style custom step temp ke pe lz pzz econserve v_dhug v_dray v_lgr_vel v_lgr_pos v_T_
->qm f_fix_id
```

It is worth noting that the temp keyword for the [*thermo_style*](#) command prints the instantaneous classical temperature T^{cl} as described by the [*fix qtb*](#) command.

2.172.5 Restrictions

This fix style is part of the QTB package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

All cell dimensions must be periodic. This fix can not be used with a triclinic cell. The QBMSST fix has been tested only for the group-ID all.

2.172.6 Related commands

fix qtb, fix msst

2.172.7 Default

The keyword defaults are $q = 10$, $\mu = 0$, $tscal = 0.01$, $damp = 1$, $seed = 880302$, $f_max = 200.0$, $N_f = 100$, $\eta = 1.0$, $\beta = 100$, and $T_init=300.0$. e_0 , p_0 , and v_0 are calculated on the first step.

(Goldman) Goldman, Reed and Fried, J. Chem. Phys. 131, 204103 (2009)

(Qi) Qi and Reed, J. Phys. Chem. A 116, 10451 (2012).

2.173 fix qeq/point command

2.174 fix qeq/shielded command

2.175 fix qeq/slater command

2.176 fix qeq/dynamic command

2.177 fix qeq/fire command

2.177.1 Syntax

`fix ID group-ID style Nevery cutoff tolerance maxiter qfile keyword ...`

- ID, group-ID are documented in [fix](#) command
- style = *qeq/point* or *qeq/shielded* or *qeq/slater* or *qeq/dynamic* or *qeq/fire*
- Nevery = perform charge equilibration every this many steps
- cutoff = global cutoff for charge-charge interactions (distance unit)
- tolerance = precision to which charges will be equilibrated
- maxiter = maximum iterations to perform charge equilibration
- qfile = a filename with QEeq parameters or *coul/streitz* or *reaxff*
- zero or more keyword/value pairs may be appended
- keyword = *alpha* or *qdamp* or *qstep* or *warn*

alpha value = Slater type orbital exponent (*qeq/slater* only)

qdamp value = damping factor for damped dynamics charge solver (*qeq/dynamic* and *qeq/fire* only)

qstep value = time step size for damped dynamics charge solver (*qeq/dynamic* and *qeq/fire* only)

warn value = do (=yes) or do not (=no) print a warning when the maximum number of iterations $\textcolor{red}{-}$ is reached

2.177.2 Examples

```
fix 1 all qeq/point 1 10 1.0e-6 200 param.qeq1
fix 1 qeq qeq/shielded 1 8 1.0e-6 100 param.qeq2
fix 1 all qeq/slater 5 10 1.0e-6 100 params alpha 0.2
fix 1 qeq qeq/dynamic 1 12 1.0e-3 100 my_qeq
fix 1 all qeq/fire 1 10 1.0e-3 100 my_qeq qdamp 0.2 qstep 0.1
```

2.177.3 Description

Perform the charge equilibration (QE_q) method as described in (*Rappe and Goddard*) and formulated in (*Nakano*) (also known as the matrix inversion method) and in (*Rick and Stuart*) (also known as the extended Lagrangian method) based on the electronegativity equilization principle.

These fixes can be used with any *pair style* in LAMMPS, so long as per-atom charges are defined. The most typical use-case is in conjunction with a *pair style* that performs charge equilibration periodically (e.g. every timestep), such as the ReaxFF or Streitz-Mintmire potential. But these fixes can also be used with potentials that normally assume per-atom charges are fixed, e.g. a *Buckingham* or *LJ/Coulombic* potential.

Because the charge equilibration calculation is effectively independent of the pair style, these fixes can also be used to perform a one-time assignment of charges to atoms. For example, you could define the QE_q fix, perform a zero-timestep run via the *run* command without any pair style defined which would set per-atom charges (based on the current atom configuration), then remove the fix via the *unfix* command before performing further dynamics.

Note

Computing and using charge values different from published values defined for a fixed-charge potential like Buckingham or CHARMM or AMBER, can have a strong effect on energies and forces, and produces a different model than the published versions.

Note

The *fix qeq/comb* command must still be used to perform charge equilibration with the *COMB potential*. The *fix qeq/reaxff* command can be used to perform charge equilibration with the *ReaxFF force field*, although *fix qeq/shielded* yields the same results as *fix qeq/reaxff* if *Nevery*, *cutoff*, and *tolerance* are the same. Eventually the *fix qeq/reaxff* command will be deprecated.

The QE_q method minimizes the electrostatic energy of the system (or equalizes the derivative of energy with respect to charge of all the atoms) by adjusting the partial charge on individual atoms based on interactions with their neighbors within *cutoff*. It requires a few parameters in the appropriate units for each atom type which are read from a file specified by *qfile*. The file has the following format

```
1 chi eta gamma zeta qcore
2 chi eta gamma zeta qcore
...
Ntype chi eta gamma zeta qcore
```

There have to be parameters given for every atom type. Wildcard entries are possible using the same type range syntax as for “coeff” commands (i.e., *n*m*, *n**, **m*, ***). Later entries will overwrite previous ones. Empty lines or any text following the pound sign (#) are ignored. Each line starts with the atom type followed by five parameters. Only a subset

of the parameters is used by each QE_Q style as described below, thus the others can be set to 0.0 if desired, but all five entries per line are required.

- *chi* = electronegativity in energy units
- *eta* = self-Coulomb potential in energy units
- *gamma* = shielded Coulomb constant defined by *ReaxFF force field* in distance units
- *zeta* = Slater type orbital exponent defined by the *Streitz-Mintmire* potential in reverse distance units
- *qcore* = charge of the nucleus defined by the *Streitz-Mintmire potential* in charge units

The fix qeq styles will print a warning if the charges are not equilibrated within *tolerance* by *maxiter* steps, unless the *warn* keyword is used with “no” as argument. This latter option may be useful for testing and benchmarking purposes, as it allows to use a fixed number of QE_Q iterations when *tolerance* is set to a small enough value to always reach the *maxiter* limit. Turning off warnings will avoid the excessive output in that case.

The *qeq/point* style describes partial charges on atoms as point charges. Interaction between a pair of charged particles is 1/r, which is the simplest description of the interaction between charges. Only the *chi* and *eta* parameters from the *qfile* file are used. Note that Coulomb catastrophe can occur if repulsion between the pair of charged particles is too weak. This style solves partial charges on atoms via the matrix inversion method. A tolerance of 1.0e-6 is usually a good number.

The *qeq/shielded* style describes partial charges on atoms also as point charges, but uses a shielded Coulomb potential to describe the interaction between a pair of charged particles. Interaction through the shielded Coulomb is given by equation (13) of the *ReaxFF force field* paper. The shielding accounts for charge overlap between charged particles at small separation. This style is the same as *fix qeq/reaxff*, and can be used with *pair_style reaxff*. Only the *chi*, *eta*, and *gamma* parameters from the *qfile* file are used. When using the string *reaxff* as filename, these parameters are extracted directly from an active *reaxff* pair style. This style solves partial charges on atoms via the matrix inversion method. A tolerance of 1.0e-6 is usually a good number.

The *qeq/slater* style describes partial charges on atoms as spherical charge densities centered around atoms via the Slater 1s orbital, so that the interaction between a pair of charged particles is the product of two Slater 1s orbitals. The expression for the Slater 1s orbital is given under equation (6) of the *Streitz-Mintmire* paper. Only the *chi*, *eta*, *zeta*, and *qcore* parameters from the *qfile* file are used. When using the string *coul/streitz* as filename, these parameters are extracted directly from an active *coul/streitz* pair style. This style solves partial charges on atoms via the matrix inversion method. A tolerance of 1.0e-6 is usually a good number. Keyword *alpha* can be used to change the Slater type orbital exponent.

The *qeq/dynamic* style describes partial charges on atoms as point charges that interact through 1/r, but the extended Lagrangian method is used to solve partial charges on atoms. Only the *chi* and *eta* parameters from the *qfile* file are used. Note that Coulomb catastrophe can occur if repulsion between the pair of charged particles is too weak. A tolerance of 1.0e-3 is usually a good number. Keyword *qdamp* can be used to change the damping factor, while keyword *qstep* can be used to change the time step size.

The **qeq/fire** style describes the same charge model and charge solver as the *qeq/dynamic* style, but employs a FIRE minimization algorithm to solve for equilibrium charges. Keyword *qdamp* can be used to change the damping factor, while keyword *qstep* can be used to change the time step size.

Note that *qeq/point*, *qeq/shielded*, and *qeq/slater* describe different charge models, whereas the matrix inversion method and the extended Lagrangian method (*qeq/dynamic* and *qeq/fire*) are different solvers.

Note that *qeq/point*, *qeq/dynamic* and *qeq/fire* styles all describe charges as point charges that interact through 1/r relationship, but solve partial charges on atoms using different solvers. These three styles should yield comparable results if the QE_Q parameters and *Nevery*, *cutoff*, and *tolerance* are the same. Style *qeq/point* is typically faster, *qeq/dynamic* scales better on larger sizes, and *qeq/fire* is faster than *qeq/dynamic*.

Note

In order to solve the self-consistent equations for electronegativity equalization, LAMMPS imposes the additional constraint that all the charges in the fix group must add up to zero. The initial charge assignments should also satisfy this constraint. LAMMPS will print a warning if that is not the case.

Note

Developing QEeq parameters (chi, eta, gamma, zeta, and qcore) is non-trivial. Charges on atoms are not guaranteed to equilibrate with arbitrary choices of these parameters. We do not develop these QEeq parameters. See the examples/qeq directory for some examples.

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

No information about these fixes is written to *binary restart files*. No global scalar or vector or per-atom quantities are stored by these fixes for access by various *output commands*. No parameter of these fixes can be used with the *start/stop* keywords of the *run* command.

These fixes are invoked during *energy minimization*.

2.177.5 Restrictions

These fixes are part of the QEeq package. They are only enabled if LAMMPS was built with that package. See the *Build package* page for more info.

These qeq fixes will ignore electric field contributions from *fix efield*.

2.177.6 Related commands

fix qeq/reaxff, *fix qeq/comb*

2.177.7 Default

warn yes

(Rappe and Goddard) A. K. Rappe and W. A. Goddard III, J Physical Chemistry, 95, 3358-3363 (1991).

(Nakano) A. Nakano, Computer Physics Communications, 104, 59-69 (1997).

(Rick and Stuart) S. W. Rick, S. J. Stuart, B. J. Berne, J Chemical Physics 101, 16141 (1994).

(Streitz-Mintmire) F. H. Streitz, J. W. Mintmire, Physical Review B, 50, 16, 11996 (1994)

(ReaxFF) A. C. T. van Duin, S. Dasgupta, F. Lorant, W. A. Goddard III, J Physical Chemistry, 105, 9396-9049 (2001)

(QEeq/Fire) T.-R. Shan, A. P. Thompson, S. J. Plimpton, in preparation

2.178 fix qeq/comb command

Accelerator Variants: *qeq/comb/omp*

2.178.1 Syntax

```
fix ID group-ID qeq/comb Nevery precision keyword value ...
```

- ID, group-ID are documented in [fix](#) command
- qeq/comb = style name of this fix command
- Nevery = perform charge equilibration every this many steps
- precision = convergence criterion for charge equilibration
- zero or more keyword/value pairs may be appended
- keyword = *file*
file value = filename
filename = name of file to write QEQ equilibration info to

2.178.2 Examples

```
fix 1 surface qeq/comb 10 0.0001
```

2.178.3 Description

Perform charge equilibration (QEQ) in conjunction with the COMB (Charge-Optimized Many-Body) potential as described in ([COMB_1](#)) and ([COMB_2](#)). It performs the charge equilibration portion of the calculation using the so-called QEeq method, whereby the charge on each atom is adjusted to minimize the energy of the system. This fix can only be used with the COMB potential; see the [fix qeq/reaxff](#) command for a QEQ calculation that can be used with any potential.

Only charges on the atoms in the specified group are equilibrated. The fix relies on the pair style (COMB in this case) to calculate the per-atom electronegativity (effective force on the charges). An electronegativity equalization calculation (or QEeq) is performed in an iterative fashion, which in parallel requires communication at each iteration for processors to exchange charge information about nearby atoms with each other. See [Rappe_and_Goddard](#) and [Rick_and_Stuart](#) for details.

During a run, charge equilibration is performed every *Nevery* time steps. Charge equilibration is also always enforced on the first step of each run. The *precision* argument controls the tolerance for the difference in electronegativity for all atoms during charge equilibration. *Precision* is a trade-off between the cost of performing charge equilibration (more iterations) and accuracy.

If the *file* keyword is used, then information about each equilibration calculation is written to the specified file.

Note

In order to solve the self-consistent equations for electronegativity equalization, LAMMPS imposes the additional constraint that all the charges in the fix group must add up to zero. The initial charge assignments should also satisfy this constraint. LAMMPS will print a warning if that is not the case.

Styles with a *gpu*, *intel*, *kk*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed on the [Accelerator packages](#) page. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the GPU, INTEL, KOKKOS, OPENMP, and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the [Build package](#) page for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the [-suffix command-line switch](#) when you invoke LAMMPS, or you can use the [suffix](#) command in your input script.

See the [Accelerator packages](#) page for more instructions on how to use the accelerated styles effectively.

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

No information about this fix is written to [binary restart files](#).

The [fix_modify respa](#) option is supported by this fix. This allows to set at which level of the [r-RESPA](#) integrator the fix is performing charge equilibration. Default is the outermost level.

This fix produces a per-atom vector which can be accessed by various [output commands](#). The vector stores the gradient of the charge on each atom. The per-atom values be accessed on any timestep.

No parameter of this fix can be used with the *start/stop* keywords of the [run](#) command.

This fix can be invoked during [energy minimization](#).

2.178.5 Restrictions

This fix command currently only supports [pair style *comb*](#).

2.178.6 Related commands

[pair_style comb](#)

2.178.7 Default

No file output is performed.

(COMB_1) J. Yu, S. B. Sinnott, S. R. Phillpot, Phys Rev B, 75, 085311 (2007),

(COMB_2) T.-R. Shan, B. D. Devine, T. W. Kemper, S. B. Sinnott, S. R. Phillpot, Phys Rev B, 81, 125328 (2010).

(Rappe_and_Goddard) A. K. Rappe, W. A. Goddard, J Phys Chem 95, 3358 (1991).

(Rick_and_Stuart) S. W. Rick, S. J. Stuart, B. J. Berne, J Chem Phys 101, 16141 (1994).

2.179 fix qeq/reaxff command

Accelerator Variants: *qeq/reaxff/kk*, *qeq/reaxff/omp*

2.179.1 Syntax

```
fix ID group-ID qeq/reaxff Nevery cutlo cuthi tolerance params args
```

- ID, group-ID are documented in [fix](#) command
- qeq/reaxff = style name of this fix command
- Nevery = perform QEeq every this many steps
- cutlo,cuthi = lo and hi cutoff for Taper radius
- tolerance = precision to which charges will be equilibrated
- params = reaxff or a filename
- one or more keywords or keyword/value pairs may be appended

keyword = dual or maxiter or nowarn

dual = process S and T matrix in parallel (only for qeq/reaxff/omp)

maxiter N = limit the number of iterations to N

nowarn = do not print a warning message if the maximum number of iterations was reached

2.179.2 Examples

```
fix 1 all qeq/reaxff 1 0.0 10.0 1.0e-6 reaxff
fix 1 all qeq/reaxff 1 0.0 10.0 1.0e-6 param.qeq maxiter 500
```

2.179.3 Description

Perform the charge equilibration (QEeq) method as described in (*Rappe and Goddard*) and formulated in (*Nakano*). It is typically used in conjunction with the ReaxFF force field model as implemented in the [pair_style reaxff](#) command, but it can be used with any potential in LAMMPS, so long as it defines and uses charges on each atom. The [fix qeq/comb](#) command should be used to perform charge equilibration with the [COMB potential](#). For more technical details about the charge equilibration performed by fix qeq/reaxff, see the (*Aktulga*) paper.

The QEeq method minimizes the electrostatic energy of the system by adjusting the partial charge on individual atoms based on interactions with their neighbors. It requires some parameters for each atom type. If the *params* setting above is the word “reaxff”, then these are extracted from the [pair_style reaxff](#) command and the ReaxFF force field file it reads in. If a file name is specified for *params*, then the parameters are taken from the specified file and the file must contain one line for each atom type. The latter form must be used when performing QeQ with a non-ReaxFF potential. Each line should be formatted as follows:

```
itype chi eta gamma
```

where *itype* is the atom type from 1 to Ntypes, *chi* denotes the electronegativity in eV, *eta* denotes the self-Coulomb potential in eV, and *gamma* denotes the valence orbital exponent. Note that these 3 quantities are also in the ReaxFF potential file, except that eta is defined here as twice the eta value in the ReaxFF file. Note that unlike the rest of LAMMPS, the units of this fix are hard-coded to be A, eV, and electronic charge.

The optional *dual* keyword allows to perform the optimization of the S and T matrices in parallel. This is only supported for the *qe/q/reaxff/omp* style. Otherwise they are processed separately. The *qe/q/reaxff/kk* style always solves the S and T matrices in parallel.

The optional *maxiter* keyword allows changing the max number of iterations in the linear solver. The default value is 200.

The optional *nowarn* keyword silences the warning message printed when the maximum number of iterations was reached. This can be useful for comparing serial and parallel results where having the same fixed number of QE_q iterations is desired, which can be achieved by using a very small tolerance and setting *maxiter* to the desired number of iterations.

Note

In order to solve the self-consistent equations for electronegativity equalization, LAMMPS imposes the additional constraint that all the charges in the fix group must add up to zero. The initial charge assignments should also satisfy this constraint. LAMMPS will print a warning if that is not the case.

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

No information about this fix is written to *binary restart files*. This fix computes a global scalar (the number of iterations) for access by various *output commands*. No parameter of this fix can be used with the *start/stop* keywords of the *run* command.

This fix is invoked during *energy minimization*.

Styles with a *gpu*, *intel*, *kk*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed on the [Accelerator packages](#) page. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the GPU, INTEL, KOKKOS, OPENMP, and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the [Build package](#) page for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the *-suffix command-line switch* when you invoke LAMMPS, or you can use the *suffix* command in your input script.

See the [Accelerator packages](#) page for more instructions on how to use the accelerated styles effectively.

2.179.5 Restrictions

This fix is part of the REAXFF package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

This fix does not correctly handle interactions involving multiple periodic images of the same atom. Hence, it should not be used for periodic cell dimensions less than 10 Angstroms.

This fix may be used in combination with *fix efield* and will apply the external electric field during charge equilibration, but there may be only one fix efield instance used and the electric field vector may only have components in non-periodic directions. Equal-style variables can be used for electric field vector components without any further settings. Atom-style variables can be used for spatially-varying electric field vector components, but the resulting electric potential must be specified as an atom-style variable using the *potential* keyword for *fix efield*.

2.179.6 Related commands

pair_style reaxff, fix qeq/shielded

2.179.7 Default

maxiter 200

(**Rappe**) Rappe and Goddard III, Journal of Physical Chemistry, 95, 3358-3363 (1991).

(**Nakano**) Nakano, Computer Physics Communications, 104, 59-69 (1997).

(**Aktulga**) Aktulga, Fogarty, Pandit, Grama, Parallel Computing, 38, 245-259 (2012).

2.180 fix qmmm command

2.180.1 Syntax

```
fix ID group-ID qmmm
```

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

2.180.2 Examples

```
fix 1 qmol qmmm
```

2.180.3 Description

This fix provides functionality to enable a quantum mechanics/molecular mechanics (QM/MM) coupling of LAMMPS to a quantum mechanical code. The current implementation only supports an ONIOM style mechanical coupling to the [Quantum ESPRESSO](#) plane wave DFT package. Electrostatic coupling is in preparation and the interface has been written in a manner that coupling to other QM codes should be possible without changes to LAMMPS itself.

The interface code for this is in the lib/qmmm directory of the LAMMPS distribution and is being made available at this early stage of development in order to encourage contributions for interfaces to other QM codes. This will allow the LAMMPS side of the implementation to be adapted if necessary before being finalized.

Details about how to use this fix are currently documented in the description of the QM/MM interface code itself in lib/qmmm/README.

2.180.4 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*.

2.180.5 Restrictions

This fix is part of the QMMM package. It is only enabled if LAMMPS was built with that package. It also requires building a library provided with LAMMPS. See the [Build package](#) page for more info.

The fix is only functional when LAMMPS is built as a library and linked with a compatible QM program and a QM/MM front end into a QM/MM executable. See the lib/qmmm/README file for details.

2.180.6 Related commands

none

2.180.7 Default

none

2.181 fix qtb command

2.181.1 Syntax

```
fix ID group-ID qtb keyword value ...
```

- ID, group-ID are documented in [fix](#) command
- qtb = style name of this fix
- zero or more keyword/value pairs may be appended
- keyword = *temp* or *damp* or *seed* or *f_max* or *N_f*
 - temp value = target quantum temperature (temperature units)
 - damp value = damping parameter (time units) inverse of friction gamma
 - seed value = random number seed (positive integer)
 - f_max value = upper cutoff frequency of the vibration spectrum (1/time units)
 - N_f value = number of frequency bins (positive integer)

2.181.2 Examples

```
# (liquid methane modeled with the REAX force field, real units)
fix 1 all nve
fix 1 all qtb temp 110 damp 200 seed 35082 f_max 0.3 N_f 100
# (quartz modeled with the BKS force field, metal units)
fix 2 all nph iso 1.01325 1.01325 1
fix 2 all qtb temp 300 damp 1 seed 47508 f_max 120.0 N_f 100
```

2.181.3 Description

This command performs the quantum thermal bath scheme proposed by ([Dammak](#)) to include self-consistent quantum nuclear effects, when used in conjunction with the [fix nve](#) or [fix nph](#) commands.

Classical molecular dynamics simulation does not include any quantum nuclear effect. Quantum treatment of the vibrational modes will introduce zero point energy into the system, alter the energy power spectrum and bias the heat capacity from the classical limit. Missing all the quantum nuclear effects, classical MD cannot model systems at temperatures lower than their classical limits. This effect is especially important for materials with a large population of hydrogen atoms and thus higher classical limits.

The equation of motion implemented by this command follows a Langevin form:

$$m_i a_i = f_i + R_i - m_i \gamma v_i$$

Here m_i , a_i , f_i , R_i , γ , and v_i represent in this order mass, acceleration, force exerted by all other atoms, random force, frictional coefficient (the inverse of damping parameter $damp$), and velocity. The random force R_i is “colored” so that any vibrational mode with frequency ω will have a temperature-sensitive energy $\theta(\omega, T)$ which resembles the energy expectation for a quantum harmonic oscillator with the same natural frequency:

$$\theta(\omega T) = \frac{\hbar}{2} + \hbar\omega \left[\exp\left(\frac{\hbar\omega}{k_B T}\right) - 1 \right]^{-1}$$

To efficiently generate the random forces, we employ the method of ([Barrat](#)), that circumvents the need to generate all random forces for all times before the simulation. The memory requirement of this approach is less demanding and independent of the simulation duration. Since the total random force R_{tot} does not necessarily vanish for a finite number of atoms, R_i is replaced by $R_i - \frac{R_{tot}}{N_{tot}}$ to avoid collective motion of the system.

The *temp* parameter sets the target quantum temperature. LAMMPS will still have an output temperature in its thermo style. That is the instantaneous classical temperature T^{cl} derived from the atom velocities at thermal equilibrium. A non-zero T^{cl} will be present even when the quantum temperature approaches zero. This is associated with zero-point energy at low temperatures.

$$T^{cl} = \sum \frac{m_i v_i^2}{3Nk_B}$$

The *damp* parameter is specified in time units, and it equals the inverse of the frictional coefficient γ . γ should be as small as possible but slightly larger than the timescale of anharmonic coupling in the system which is about 10 ps to 100 ps. When γ is too large, it gives an energy spectrum that differs from the desired Bose-Einstein spectrum. When γ is too small, the quantum thermal bath coupling to the system will be less significant than anharmonic effects, reducing to a classical limit. We find that setting γ between 5 THz and 1 THz could be appropriate depending on the system.

The random number *seed* is a positive integer used to initiate a Marsaglia random number generator. Each processor uses the input seed to generate its own unique seed and its own stream of random numbers. Thus the dynamics of the system will not be identical on two runs on different numbers of processors.

The *f_max* parameter truncates the noise frequency domain so that vibrational modes with frequencies higher than *f_max* will not be modulated. If we denote Δt as the time interval for the MD integration, *f_max* is always reset by the code to

make $\alpha = (\text{int})(2f_{\text{max}} \Delta t)^{-1}$ a positive integer and print out relative information. An appropriate value for the cutoff frequency f_{max} would be around 2~3 f_D , where f_D is the Debye frequency.

The N_f parameter is the frequency grid size, the number of points from 0 to f_{max} in the frequency domain that will be sampled. $3*2N_f$ per-atom random numbers are required in the random force generation and there could be as many atoms as in the whole simulation that can migrate into every individual processor. A larger N_f provides a more accurate sampling of the spectrum while consumes more memory. With fixed f_{max} and γ , N_f should be big enough to converge the classical temperature T^{cl} as a function of target quantum bath temperature. Memory usage per processor could be from 10 to 100 MBytes.

Note

Unlike the [fix nvt](#) command which performs Nose/Hoover thermostatting AND time integration, this fix does NOT perform time integration. It only modifies forces to a colored thermostat. Thus you must use a separate time integration fix, like [fix nve](#) or [fix nph](#) to actually update the velocities and positions of atoms (as shown in the examples). Likewise, this fix should not normally be used with other fixes or commands that also specify system temperatures , e.g. [fix nvt](#) and [fix temp/rescale](#).

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

No information about this fix is written to [binary restart files](#). Because the state of the random number generator is not saved in restart files, this means you cannot do “exact” restarts with this fix. However, in a statistical sense, a restarted simulation should produce similar behaviors of the system.

This fix is not invoked during [energy minimization](#).

2.181.5 Restrictions

This fix style is part of the QTB package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

2.181.6 Related commands

[fix nve](#), [fix nph](#), [fix langevin](#), [fix qbmsst](#)

2.181.7 Default

The keyword defaults are temp = 300, damp = 1, seed = 880302, f_max=200.0 and N_f = 100.

(Dammak) Dammak, Chalopin, Laroche, Hayoun, and Greffet, Phys Rev Lett, 103, 190601 (2009).

(Barrat) Barrat and Rodney, J. Stat. Phys, 144, 679 (2011).

2.182 fix reaxff/bonds command

Accelerator Variants: *reaxff/bonds/kk*

2.182.1 Syntax

```
fix ID group-ID reaxff/bonds Nevery filename
```

- ID, group-ID are documented in [fix](#) command
- reax/bonds = style name of this fix command
- Nevery = output interval in timesteps
- filename = name of output file

2.182.2 Examples

```
fix 1 all reaxff/bonds 100 bonds.reaxff
```

2.182.3 Description

Write out the bond information computed by the ReaxFF potential specified by [pair style reaxff](#) in the exact same format as the original stand-alone ReaxFF code of Adri van Duin. The bond information is written to *filename* on timesteps that are multiples of *Nevery*, including timestep 0. For time-averaged chemical species analysis, please see the [fix reaxff/species](#) command.

The specified group-ID is ignored by this fix.

The format of the output file should be reasonably self-explanatory. The meaning of the column header abbreviations is as follows:

- id = atom id
- type = atom type
- nb = number of bonds
- id_1 = atom id of first bond
- id_nb = atom id of Nth bond
- mol = molecule id
- bo_1 = bond order of first bond
- bo_nb = bond order of Nth bond
- abo = atom bond order (sum of all bonds)
- nlp = number of lone pairs
- q = atomic charge

If the filename ends with “.gz”, the output file 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.

2.182.4 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](#).

Styles with a *gpu*, *intel*, *kk*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed on the [Accelerator packages](#) page. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the GPU, INTEL, KOKKOS, OPENMP, and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the [Build package](#) page for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the [-suffix command-line switch](#) when you invoke LAMMPS, or you can use the [suffix](#) command in your input script.

See the [Accelerator packages](#) page for more instructions on how to use the accelerated styles effectively.

2.182.5 Restrictions

The fix reaxff/bonds command requires that the [pair_style reaxff](#) is invoked. This fix is part of the REAXFF package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

To write gzipped bond files, you must compile LAMMPS with the -DLAMMPS_GZIP option.

2.182.6 Related commands

[pair_style reaxff](#), [fix reaxff/species](#)

2.182.7 Default

none

2.183 fix reaxff/species command

Accelerator Variants: *reaxff/species/kk*

2.183.1 Syntax

```
fix ID group-ID reaxff/species Nevery Nrepeat Nfreq filename keyword value ...
```

- ID, group-ID are documented in [fix](#) command
- reaxff/species = style name of this command
- Nevery = sample bond-order every this many timesteps
- Nrepeat = # of bond-order samples used for calculating averages

- `Nfreq` = calculate average bond-order every this many timesteps
- `filename` = name of output file
- zero or more keyword/value pairs may be appended
- keyword = `cutoff` or `element` or `position` or `delete` or `delete_rate_limit`

cutoff value = I J Cutoff

I, J = atom types (see asterisk form below)

Cutoff = Bond-order cutoff value for this pair of atom types

element value = Element1, Element2, ...

position value = posfreq filepos

posfreq = write position files every this many timestep

filepos = name of position output file

delete value = filedel keyword value

filedel = name of delete species output file

keyword = specieslist or masslimit

specieslist value = Nspecies Species1 Species2 ...

Nspecies = number of species in list

masslimit value = massmin massmax

massmin = minimum molecular weight of species to delete

massmax = maximum molecular weight of species to delete

delete_rate_limit value = Nlimit Nsteps

Nlimit = maximum number of deletions allowed to occur within interval

Nsteps = the interval (number of timesteps) over which to count deletions

2.183.2 Examples

```
fix 1 all reaxff/species 10 10 100 species.out
fix 1 all reaxff/species 1 2 20 species.out cutoff 1 1 0.40 cutoff 1 2*3 0.55
fix 1 all reaxff/species 1 100 100 species.out element Au O H position 1000 AuOH.pos
fix 1 all reaxff/species 1 100 100 species.out delete species.del masslimit 0 50
```

2.183.3 Description

Write out the chemical species information computed by the ReaxFF potential specified by `pair_style reaxff`. Bond-order values (either averaged or instantaneous, depending on value of `Nrepeat`) are used to determine chemical bonds. Every `Nfreq` timesteps, chemical species information is written to `filename` as a two line output. The first line is a header containing labels. The second line consists of the following: timestep, total number of molecules, total number of distinct species, number of molecules of each species. In this context, “species” means a unique molecule. The chemical formula of each species is given in the first line.

Warning

In order to compute averaged data, it is required that there are no neighbor list rebuilds for at least `Nrepeat*Nevery` steps preceding each `Nfreq` step. For that reason, fix `reaxff/species` may change your neighbor list settings. Renighboring will occur no more frequently than every `Nrepeat*Nevery` timesteps, and will occur less frequently if `Nfreq` is not a multiple of `Nrepeat*Nevery`. There will be a warning message showing the new settings. Having a `Nfreq` setting that is larger than what is required for correct computation of the ReaxFF force field interactions, in combination with certain `Nrepeat` and `Nevery` settings, can thus lead to incorrect results. For typical ReaxFF calculations, renighboring only every 100 steps is already quite a low frequency.

If the filename ends with “.gz”, the output file 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.

Added in version 15Jun2023: Support for wildcards added

Optional keyword *cutoff* can be assigned to change the minimum bond-order values used in identifying chemical bonds between pairs of atoms. Bond-order cutoffs should be carefully chosen, as bond-order cutoffs that are too small may include too many bonds (which will result in an error), while cutoffs that are too large will result in fragmented molecules. The default cutoff of 0.3 usually gives good results. A wildcard asterisk can be used in place of or in conjunction with the I,J arguments to set the bond-order cutoff for multiple pairs of atom types. This takes the form “*” or “*n” or “n*” or “m*n”. If *N* is 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 optional keyword *element* can be used to specify the chemical symbol printed for each LAMMPS atom type. The number of symbols must match the number of LAMMPS atom types and each symbol must consist of 1 or 2 alphanumeric characters. By default, these symbols are the same as the chemical identity of each LAMMPS atom type, as specified by the [ReaxFF pair_coeff](#) command and the ReaxFF force field file.

The optional keyword *position* writes center-of-mass positions of each identified molecules to file *filepos* every *posfreq* timesteps. The first line contains information on timestep, total number of molecules, total number of distinct species, and box dimensions. The second line is a header containing labels. From the third line downward, each molecule writes a line of output containing the following information: molecule ID, number of atoms in this molecule, chemical formula, total charge, and center-of-mass xyz positions of this molecule. The xyz positions are in fractional coordinates relative to the box dimensions.

For the keyword *position*, the *filepos* is the name of the output file. It can contain the wildcard character “*”. If the “*” character appears in *filepos*, then one file per snapshot is written at *posfreq* and the “*” character is replaced with the timestep value. For example, AuO.pos.* becomes AuO.pos.0, AuO.pos.1000, etc.

Added in version 3Aug2022.

The optional keyword *delete* enables the periodic removal of molecules from the system ([Gissinger](#)). Criteria for deletion can be either a list of specific chemical formulae or a range of molecular weights. Molecules are deleted every *Nfreq* timesteps, and bond connectivity is determined using the *Nevery* and *Nrepeat* keywords. The *filedel* argument is the name of the output file that records the species that are removed from the system. The *specieslist* keyword permits specific chemical species to be deleted. The *Nspecies* argument specifies how many species are eligible for deletion and is followed by a list of chemical formulae, whose strings are compared to species identified by this fix. For example, “specieslist 2 CO CO2” deletes molecules that are identified as “CO” and “CO2” in the species output file. When using the *specieslist* keyword, the *filedel* file has the following format: the first line lists the chemical formulae eligible for deletion, and each additional line contains the timestep on which a molecule deletion occurs and the number of each species deleted on that timestep. The *masslimit* keyword permits deletion of molecules with molecular weights between *massmin* and *massmax*. When using the *masslimit* keyword, each line of the *filedel* file contains the timestep on which deletions occurs, followed by how many of each species are deleted (with quantities preceding chemical formulae). The *specieslist* and *masslimit* keywords cannot both be used in the same *reaxff/species* fix. The *delete_rate_limit* keyword can enforce an upper limit on the overall rate of molecule deletion. The number of deletion occurrences is limited to *Nlimit* within an interval of *Nsteps* timesteps. *Nlimit* can be specified with an equal-style [variable](#). When using the *delete_rate_limit* keyword, no deletions are permitted to occur within the first *Nsteps* timesteps of the first run (after reading a either a data or restart file).

The *Nevery*, *Nrepeat*, and *Nfreq* arguments specify on what timesteps the bond-order values are sampled to get the average bond order. The species analysis is performed using the average bond-order on timesteps that are a multiple of *Nfreq*. The average is over *Nrepeat* bond-order samples, 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 bond-order cannot overlap, i.e. *Nrepeat*Nevery* can not exceed *Nfreq*.

For example, if Nevery=2, Nrepeat=6, and Nfreq=100, then bond-order values on timesteps 90,92,94,96,98,100 will be used to compute the average bond-order for the species analysis output on timestep 100.

2.183.4 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 both a global vector of length 2 and a per-atom vector, either of which can be accessed by various *output commands*. The values in the global vector are “intensive”.

The 2 values in the global vector are as follows:

1. total number of molecules
2. total number of distinct species

The per-atom vector stores the molecule ID for each atom as identified by the fix. If an atom is not in a molecule, its ID will be 0. For atoms in the same molecule, the molecule ID for all of them will be the same and will be equal to the smallest atom ID of any atom in the molecule.

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*.

Styles with a *gpu*, *intel*, *kk*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed on the *Accelerator packages* page. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the GPU, INTEL, KOKKOS, OPENMP, and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the *Build package* page for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the *-suffix command-line switch* when you invoke LAMMPS, or you can use the *suffix* command in your input script.

See the *Accelerator packages* page for more instructions on how to use the accelerated styles effectively.

2.183.5 Restrictions

The “fix reaxff/species” requires that *pair_style reaxff* is used. This fix is part of the REAXFF package. It is only enabled if LAMMPS was built with that package. See the *Build package* page for more info.

To write gzipped species files, you must compile LAMMPS with the *-DLAMMPS_GZIP* option.

2.183.6 Related commands

pair_style reaxff, *fix reaxff/bonds*

2.183.7 Default

The default values for bond-order cutoffs are 0.3 for all I-J pairs. The default element symbols are taken from the ReaxFF pair_coeff command. Position files are not written by default.

(Gissinger) Jacob R. Gissinger, Scott R. Zavada, Joseph G. Smith, Josh Kemppainen, Ivan Gallegos, Gregory M. Odegard, Emilie J. Siochi, and Kristopher E. Wise, Carbon, 202, 336-347 (2023).

2.184 fix recenter command

2.184.1 Syntax

```
fix ID group-ID recenter x y z keyword value ...
```

- ID, group-ID are documented in [fix](#) command
- recenter = style name of this fix command
- x,y,z = constrain center-of-mass to these coords (distance units), any coord can also be NULL or INIT (see below)
- zero or more keyword/value pairs may be appended
- keyword = *shift* or *units*
 - shift value = group-ID
 - group-ID = group of atoms whose coords are shifted
 - units value = box or lattice or fraction

2.184.2 Examples

```
fix 1 all recenter 0.0 0.5 0.0
fix 1 all recenter INIT INIT NULL
fix 1 all recenter INIT 0.0 0.0 units box
```

2.184.3 Description

Constrain the center-of-mass position of a group of atoms by adjusting the coordinates of the atoms every timestep. This is simply a small shift that does not alter the dynamics of the system or change the relative coordinates of any pair of atoms in the group. This can be used to ensure the entire collection of atoms (or a portion of them) do not drift during the simulation due to random perturbations (e.g. [fix langevin](#) thermostating).

Distance units for the x,y,z values are determined by the setting of the *units* keyword, as discussed below. One or more x,y,z values can also be specified as NULL, which means exclude that dimension from this operation. Or it can be specified as INIT which means to constrain the center-of-mass to its initial value at the beginning of the run.

The center-of-mass (COM) is computed for the group specified by the fix. If the current COM is different than the specified x,y,z, then a group of atoms has their coordinates shifted by the difference. By default the shifted group is also the group specified by the fix. A different group can be shifted by using the *shift* keyword. For example, the COM could be computed on a protein to keep it in the center of the simulation box. But the entire system (protein + water) could be shifted.

If the *units* keyword is set to *box*, then the distance units of x,y,z are defined by the [units](#) command - e.g. Angstroms for *real* units. A *lattice* value means the distance units are in lattice spacings. The [lattice](#) command must have been previously used to define the lattice spacing. A *fraction* value means a fractional distance between the lo/hi box boundaries, e.g. 0.5 = middle of the box. The default is to use lattice units.

Note that the [velocity](#) command can be used to create velocities with zero aggregate linear and/or angular momentum.

Note

This fix performs its operations at the same point in the timestep as other time integration fixes, such as [fix nve](#), [fix nvt](#), or [fix npt](#). Thus fix recenter should normally be the last such fix specified in the input script, since the adjustments it makes to atom coordinates should come after the changes made by time integration. LAMMPS will warn you if your fixes are not ordered this way.

Note

If you use this fix on a small group of atoms (e.g. a molecule in solvent) without using the *shift* keyword to adjust the positions of all atoms in the system, then the results can be unpredictable. For example, if the molecule is pushed consistently in one direction by a flowing solvent, its velocity will increase. But its coordinates will be re-centered, meaning it is moved back towards the force. Thus over time, the velocity and effective temperature of the molecule could become very large, though it won't actually be moving due to the re-centering. If you are thermostating the entire system, then the solvent would be cooled to compensate. A better solution for this simulation scenario is to use the [fix spring](#) command to tether the molecule in place.

2.184.4 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 distance the group is moved by fix recenter.

This fix also computes global 3-vector which can be accessed by various [output commands](#). The 3 quantities in the vector are xyz components of displacement applied to the group of atoms by the fix.

The scalar and vector values calculated by this fix are “extensive”.

No parameter of this fix can be used with the *start/stop* keywords of the [run](#) command. This fix is not invoked during [energy minimization](#).

2.184.5 Restrictions

This fix should not be used with an x,y,z setting that causes a large shift in the system on the first timestep, due to the requested COM being very different from the initial COM. This could cause atoms to be lost, especially in parallel. Instead, use the [displace_atoms](#) command, which can be used to move atoms a large distance.

2.184.6 Related commands

fix momentum, velocity

2.184.7 Default

The option defaults are shift = fix group-ID, and units = lattice.

2.185 fix restrain command

2.185.1 Syntax

```
fix ID group-ID restrain keyword args ...
```

- ID, group-ID are documented in [fix](#) command
- restrain = style name of this fix command
- one or more keyword/arg pairs may be appended
- keyword = *bond* or *lbound* or *angle* or *dihedral*

bond args = atom1 atom2 Kstart Kstop r0start (r0stop)
atom1,atom2 = IDs of two atoms in bond
Kstart,Kstop = restraint coefficients at start/end of run (energy units)
r0start = equilibrium bond distance at start of run (distance units)
r0stop = equilibrium bond distance at end of run (optional) (distance units). If not specified it is assumed to be equal to r0start
lbound args = atom1 atom2 Kstart Kstop r0start (r0stop)
atom1,atom2 = IDs of two atoms in bond
Kstart,Kstop = restraint coefficients at start/end of run (energy units)
r0start = equilibrium bond distance at start of run (distance units)
r0stop = equilibrium bond distance at end of run (optional) (distance units). If not specified it is assumed to be equal to r0start
angle args = atom1 atom2 atom3 Kstart Kstop theta0
atom1,atom2,atom3 = IDs of three atoms in angle, atom2 = middle atom
Kstart,Kstop = restraint coefficients at start/end of run (energy units)
theta0 = equilibrium angle theta (degrees)
dihedral args = atom1 atom2 atom3 atom4 Kstart Kstop phi0 keyword/value
atom1,atom2,atom3,atom4 = IDs of 4 atoms in dihedral in linear order
Kstart,Kstop = restraint coefficients at start/end of run (energy units)
phi0 = equilibrium dihedral angle phi (degrees)
keyword/value = optional keyword value pairs. supported keyword/value pairs:
mult n = dihedral multiplicity n (integer ≥ 0 , default = 1)

2.185.2 Examples

```
fix holdem all restrain bond 45 48 2000.0 2000.0 2.75
fix holdem all restrain lbound 45 48 2000.0 2000.0 2.75
fix holdem all restrain dihedral 1 2 3 4 2000.0 2000.0 120.0
fix holdem all restrain bond 45 48 2000.0 2000.0 2.75 dihedral 1 2 3 4 2000.0 2000.0 120.0
fix texas_holdem all restrain dihedral 1 2 3 4 0.0 2000.0 120.0 dihedral 1 2 3 5 0.0 2000.0 -120.0 dihedral_
→ 1 2 3 6 0.0 2000.0 0.0
```

2.185.3 Description

Restrain the motion of the specified sets of atoms by making them part of a bond or angle or dihedral interaction whose strength can vary over time during a simulation. This is functionally similar to creating a bond or angle or dihedral for the same atoms in a data file, as specified by the [read_data](#) command, albeit with a time-varying prefactor coefficient, and except for exclusion rules, as explained below.

For the purpose of force field parameter-fitting or mapping a molecular potential energy surface, this fix reduces the hassle and risk associated with modifying data files. In other words, use this fix to temporarily force a molecule to adopt a particular conformation. To create a permanent bond or angle or dihedral, you should modify the data file.

Note

Adding a bond/angle/dihedral with this command does not apply the exclusion rules and weighting factors specified by the [special_bonds](#) command to atoms in the restraint that are now bonded (1-2,1-3,1-4 neighbors) as a result. If they are close enough to interact in a [pair_style](#) sense (non-bonded interaction), then the bond/angle/dihedral restraint interaction will simply be superposed on top of that interaction.

The group-ID specified by this fix is ignored.

The second example above applies a restraint to hold the dihedral angle formed by atoms 1, 2, 3, and 4 near 120 degrees using a constant restraint coefficient. The fourth example applies similar restraints to multiple dihedral angles using a restraint coefficient that increases from 0.0 to 2000.0 over the course of the run.

Note

Adding a force to atoms implies a change in their potential energy as they move due to the applied force field. For dynamics via the [run](#) command, this energy can be added to the system's potential energy for thermodynamic output (see below). For energy minimization via the [minimize](#) command, this energy must be added to the system's potential energy to formulate a self-consistent minimization problem (see below).

In order for a restraint to be effective, the restraint force must typically be significantly larger than the forces associated with conventional force field terms. If the restraint is applied during a dynamics run (as opposed to during an energy minimization), a large restraint coefficient can significantly reduce the stable timestep size, especially if the atoms are initially far from the preferred conformation. You may need to experiment to determine what value of K works best for a given application.

For the case of finding a minimum energy structure for a single molecule with particular restraints (e.g. for fitting force field parameters or constructing a potential energy surface), commands such as the following may be useful:

```
# minimize molecule energy with restraints
velocity all create 600.0 8675309 mom yes rot yes dist gaussian
```

(continues on next page)

(continued from previous page)

```

fix NVE all nve
fix TFIIX all langevin 600.0 0.0 100 24601
fix REST all restrain dihedral 2 1 3 8 0.0 5000.0 ${angle1} dihedral 3 1 2 9 0.0 5000.0 ${angle2}
fix_modify REST energy yes
run 10000
fix TFIIX all langevin 0.0 0.0 100 24601
fix REST all restrain dihedral 2 1 3 8 5000.0 5000.0 ${angle1} dihedral 3 1 2 9 5000.0 5000.0 ${angle2}
fix_modify REST energy yes
run 10000
# sanity check for convergence
minimize 1e-6 1e-9 1000 100000
# report unrestrained energies
unfix REST
run 0

```

The *bond* keyword applies a bond restraint to the specified atoms using the same functional form used by the *bond_style harmonic* command. The potential associated with the restraint is

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

with the following coefficients:

- K (energy/distance²)
- r_0 (distance)

K and r_0 are specified with the fix. Note that the usual 1/2 factor is included in K .

The *lbound* keyword applies a lower bound bond restraint to the specified atoms using the same functional form used by the *bond_style harmonic* command if the distance between the atoms is smaller than the equilibrium bond distance and 0 otherwise. The potential associated with the restraint is

$$E = K(r - r_0)^2, \text{ if } r < r_0$$

$$E = 0 \quad , \text{ if } r \geq r_0$$

with the following coefficients:

- K (energy/distance²)
- r_0 (distance)

K and r_0 are specified with the fix. Note that the usual 1/2 factor is included in K .

The *angle* keyword applies an angle restraint to the specified atoms using the same functional form used by the *angle_style harmonic* command. The potential associated with the restraint is

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

with the following coefficients:

- K (energy)
- θ_0 (degrees)

K and θ_0 are specified with the fix. θ_0 is specified in degrees, but LAMMPS converts it to radians internally; hence K is effectively energy per radian². Note that the usual 1/2 factor is included in K .

The *dihedral* keyword applies a dihedral restraint to the specified atoms using a simplified form of the function used by the *dihedral_style charmm* command. The potential associated with the restraint is

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

with the following coefficients:

- K (energy)
- n (multiplicity, $>= 0$)
- d (degrees) = $\phi_0 + 180$

K and ϕ_0 are specified with the fix. Note that the value of the dihedral multiplicity n is set by default to 1. You can use the optional *mult* keyword to set it to a different positive integer. Also note that the energy will be a minimum when the current dihedral angle ϕ is equal to ϕ_0 .

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

No information about this fix is written to *binary restart files*.

The *fix_modify energy* option is supported by this fix to add the potential energy associated with this fix to the global potential energy of the system as part of *thermodynamic output*. The default setting for this fix is *fix_modify energy no*.

The *fix_modify respa* option is supported by this fix. This allows to set at which level of the *r-RESPA* integrator the fix is adding its forces. Default is the outermost level.

Note

If you want the fictitious potential energy associated with the added forces to be included in the total potential energy of the system (the quantity being minimized), you MUST enable the *fix_modify energy* option for this fix.

This fix computes a global scalar and a global vector of length 3, which can be accessed by various *output commands*. The scalar is the total potential energy for *all* the restraints as discussed above. The vector values are the sum of contributions to the following individual categories:

1. bond energy
2. angle energy
3. dihedral energy

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.

2.185.5 Restrictions

none

2.185.6 Related commands

none

2.185.7 Default

none

2.186 fix rheo command

2.186.1 Syntax

```
fix ID group-ID rheo cut kstyle zmin keyword values...
```

- ID, group-ID are documented in [fix](#) command
- rheo = style name of this fix command
- cut = cutoff for the kernel (distance)
- kstyle = *quintic* or *RK0* or *RK1* or *RK2*
- zmin = minimal number of neighbors for reproducing kernels
- zero or more keyword/value pairs may be appended to args
- **keyword** = *thermal* or *interface/reconstruct* or *surface/detection* or
shift or *rho/sum* or *density* or *self/mass* or *speed/sound*

thermal values = none, turns on thermal evolution

interface/reconstruct values = none, reconstructs interfaces with solid particles

surface/detection values = sdstyle limit limit/splash

sdstyle = coordination or divergence

limit = threshold for surface particles

limit/splash = threshold for splash particles

shift values = none, turns on velocity shifting

rho/sum values = none, uses the kernel to compute the density of particles

self/mass values = none, a particle uses its own mass in a rho summation

density values = rho01, ... rho0N (density)

speed/sound values = cs0, ... csN (velocity)

2.186.2 Examples

```
fix 1 all rheo 3.0 quintic 0 thermal density 0.1 0.1 speed/sound 10.0 1.0
fix 1 all rheo 3.0 RK1 10 shift surface/detection coordination 40
```

2.186.3 Description

Added in version 29Aug2024.

Perform time integration for RHEO particles, updating positions, velocities, and densities. For an overview of other features available in the RHEO package, see [the RHEO howto](#).

The type of kernel is specified using *kstyle* and the cutoff is *cut*. Four kernels are currently available. The *quintic* kernel is a standard quintic spline function commonly used in SPH. The other options, *RK0*, *RK1*, and *RK2*, are zeroth, first, and second order reproducing. To generate a reproducing kernel, a particle must have sufficient neighbors inside the kernel cutoff distance (a coordination number) to accurately calculate moments. This threshold is set by *zmin*. If reproducing kernels are requested but a particle has fewer neighbors, then it will revert to a non-reproducing quintic kernel until it gains more neighbors.

To model temperature evolution, one must specify the *thermal* keyword, define a separate instance of *fix rheo/thermal*, and use atom style *rheo/thermal*.

By default, the density of solid RHEO particles does not evolve and forces with fluid particles are calculated using the current velocity of the solid particle. If the *interface/reconstruct* keyword is used, then the density and velocity of solid particles are alternatively reconstructed for every fluid-solid interaction to ensure no-slip and pressure-balanced boundaries. This is done by estimating the location of the fluid-solid interface and extrapolating fluid particle properties across the interface to calculate a temporary apparent density and velocity for a solid particle.

A modified form of Fickian particle shifting can be enabled with the *shift* keyword. This effectively shifts particle positions to generate a more uniform spatial distribution. Shifting currently does not consider the type of a particle and therefore may be inappropriate in systems consisting of multiple fluid phases.

In systems with free surfaces, the *surface/detection* keyword can be used to classify the location of particles as being within the bulk fluid, on a free surface, or isolated from other particles in a splash or droplet. Shifting is then disabled in the normal direction away from the free surface to prevent particles from diffusing away. Surface detection can also be used to control surface-nucleated effects like oxidation when used in combination with *fix rheo/oxidation*. Surface detection is not performed on solid bodies.

The *surface/detection* keyword takes three arguments: *sdstyle*, *limit*, and *limit/splash*. The first, *sdstyle*, specifies whether surface particles are identified using a coordination number (*coordination*) or the divergence of the local particle positions (*divergence*). The threshold value for a surface particle for either of these criteria is set by the numerical value of *limit*. Additionally, if a particle's coordination number is too low, i.e. if it has separated off from the bulk in a droplet, it is not possible to define surfaces and the particle is classified as a splash. The coordination threshold for this classification is set by the numerical value of *limit/splash*.

By default, RHEO integrates particles' densities using a mass diffusion equation. Alternatively, one can update densities every timestep by performing a kernel summation of the masses of neighboring particles by specifying the *rho/sum* keyword.

The *self/mass* keyword modifies the behavior of the density summation in *rho/sum*. Typically, the density ρ of a particle is calculated as the sum over neighbors

$$\rho_i = \sum_j W_{ij} M_j$$

where W_{ij} is the kernel, and M_j is the mass of particle j . The *self/mass* keyword augments this expression by replacing M_j with M_i . This may be useful in simulations of multiple fluid phases with large differences in density, (*Hu*).

The *density* keyword is used to specify the equilibrium density of each of the N particle types. It must be followed by N numerical values specifying each type's equilibrium density *rho0*.

The *speed/sound* keyword is used to specify the speed of sound of each of the N particle types. It must be followed by N numerical values specifying each type's speed of sound *cs*.

2.186.4 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*.

2.186.5 Restrictions

This fix must be used with atom style rheo or rheo/thermal. This fix must be used in conjunction with *fix rheo/pressure* and *fix rheo/viscosity*. If the *thermal* setting is used, there must also be an instance of *fix rheo/thermal*. The fix group must be set to all. Only one instance of fix rheo may be defined and it must be defined prior to all other RHEO fixes in the input script.

This fix is part of the RHEO package. It is only enabled if LAMMPS was built with that package. See the *Build package* page for more info.

2.186.6 Related commands

fix rheo/viscosity, *fix rheo/pressure*, *fix rheo/thermal*, *pair rheo*, *compute rheo/property/atom*

2.186.7 Default

rho0 and *cs* are set to 1.0 for all atom types.

(Hu) Hu, and Adams J. Comp. Physics, 213, 844-861 (2006).

2.187 fix rheo/oxidation command

2.187.1 Syntax

```
fix ID group-ID rheo/oxidation cut btype rsurf
```

- ID, group-ID are documented in *fix* command
- rheo/oxidation = style name of this fix command
- cut = maximum bond length (distance units)
- btype = type of bonds created
- rsurf = distance from surface to create bonds (distance units)

2.187.2 Examples

```
fix 1 all rheo/oxidation 1.5 2 0.0
fix 1 all rheo/oxidation 1.0 1 2.0
```

2.187.3 Description

Added in version 29Aug2024.

This fix dynamically creates bonds on the surface of fluids to represent physical processes such as oxidation. It is intended for use with bond style [bond rheo/shell](#).

Every timestep, particles check neighbors within a distance of *cut*. This distance must be smaller than the kernel length defined in [fix rheo](#). Bonds of type *btype* are created between a fluid particle and either a fluid or solid neighbor. The fluid particles must also be on the fluid surface, or within a distance of *rsurf* from the surface. This process is further described in ([Clemmer](#)).

If used in conjunction with solid bodies, such as those generated by the *react* option of [fix rheo/thermal](#), it is recommended to use a [hybrid bond style](#) with different bond types for solid and oxide bonds.

2.187.4 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](#).

2.187.5 Restrictions

This fix must be used with the bond style [rheo/shell](#) and [fix rheo](#) with surface detection enabled.

This fix is part of the RHEO package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

2.187.6 Related commands

[fix rheo](#), [bond rheo/shell](#), [compute rheo/property/atom](#)

2.187.7 Default

none

([Clemmer](#)) Clemmer, Pierce, O'Connor, Nevins, Jones, Lechman, Tencer, Appl. Math. Model., 130, 310-326 (2024).

2.188 fix rheo/pressure command

2.188.1 Syntax

```
fix ID group-ID rheo/pressure type1 pstyle1 args1 ... typeN pstyleN argsN
```

- ID, group-ID are documented in [fix](#) command
- rheo/pressure = style name of this fix command
- one or more types and pressure styles must be appended
- types = lists of types (see below)
- pstyle = *linear* or *taitwater* or *cubic*
 - linear* args = none
 - taitwater* args = none
 - cubic* args = cubic prefactor A_3 (pressure/density²)

2.188.2 Examples

```
fix 1 all rheo/pressure * linear
fix 1 all rheo/pressure 1 linear 2 cubic 10.0
```

2.188.3 Description

Added in version 29Aug2024.

This fix defines a pressure equation of state for RHEO particles. One can define different equations of state for different atom types. An equation must be specified for every atom type.

One first defines the atom *types*. A wild-card asterisk can be used in place of or in conjunction with the *types* argument to set the coefficients for multiple pairs of atom types. This takes the form “*” or “*n” or “m*” or “m*n”. If N is 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 m to N (inclusive). A middle asterisk means all types from m to n (inclusive).

The *types* definition is followed by the pressure style, *pstyle*. Current options *linear*, *taitwater*, and *cubic*. Style *linear* is a linear equation of state with a particle pressure P calculated as

$$P = c(\rho - \rho_0)$$

where c is the speed of sound, ρ_0 is the equilibrium density, and ρ is the current density of a particle. The numerical values of c and ρ_0 are set in [fix rheo](#). Style *cubic* is a cubic equation of state which has an extra argument A_3 ,

$$P = c((\rho - \rho_0) + A_3(\rho - \rho_0)^3).$$

Style *taitwater* is Tait's equation of state:

$$P = \frac{c^2 \rho_0}{7} \left[\left(\frac{\rho}{\rho_0} \right)^7 - 1 \right].$$

2.188.4 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](#).

2.188.5 Restrictions

This fix must be used with an atom style that includes density such as atom_style rheo or rheo/thermal. This fix must be used in conjunction with [fix rheo](#). The fix group must be set to all. Only one instance of fix rheo/pressure can be defined.

This fix is part of the RHEO package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

2.188.6 Related commands

[fix rheo](#), [pair rheo](#), [compute rheo/property/atom](#)

2.188.7 Default

none

2.189 fix rheo/thermal command

2.189.1 Syntax

fix ID group-ID rheo/thermal attribute values ...

- ID, group-ID are documented in [fix](#) command
 - rheo/thermal = style name of this fix command
 - one or more attributes may be appended
 - attribute = *conductivity* or *specific/heat* or *latent/heat* or *Tfreeze* or *react*
- conductivity args = types style args
 types = lists of types (see below)
 style = constant
 constant arg = conductivity (power/temperature)
- specific/heat args = types style args
 types = lists of types (see below)
 style = constant
 constant arg = specific heat (energy/(mass*temperature))
- latent/heat args = types style args
 types = lists of types (see below)
 style = constant
 constant arg = latent heat (energy/mass)
- Tfreeze args = types style args
 types = lists of types (see below)
 style = constant

constant arg = freezing temperature (temperature)
 react args = cut type
 cut = maximum bond distance
 type = bond type

2.189.2 Examples

```
fix 1 all rheo/thermal conductivity * constant 1.0 specific/heat * constant 1.0 Tfreeze * constant 1.0
fix 1 all rheo/pressure conductivity 1*2 constant 1.0 conductivity 3*4 constant 2.0 specific/heat *
  ↪constant 1.0
```

2.189.3 Description

Added in version 29Aug2024.

This fix performs time integration of temperature for atom style rheo/thermal. In addition, it defines multiple thermal properties of particles and handles melting/solidification, if applicable. For more details on phase transitions in RHEO, see [the RHEO howto](#).

Note that the temperature of a particle is always derived from the energy. This implies the *temperature* attribute of [the set command](#) does not affect particles. Instead, one should use the *sph/e* attribute.

For each atom type, one can define expressions for the *conductivity*, *specific/heat*, *latent/heat*, and critical temperature (*Tfreeze*). The conductivity and specific heat must be defined for all atom types. The latent heat and critical temperature are optional. However, a critical temperature must be defined to specify a latent heat.

Note, if shifting is turned on in [fix rheo](#), the gradient of the energy is used to shift energies. This may be inappropriate in systems with multiple atom types with different specific heats.

For each property, one must first define a list of atom types. A wild-card asterisk can be used in place of or in conjunction with the *types* argument to set the coefficients for multiple pairs of atom types. This takes the form “*” or “*n” or “m*” or “m*n”. If *N* is 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 *m* to *N* (inclusive). A middle asterisk means all types from *m* to *n* (inclusive).

The *types* definition for each property is followed by the style. Currently, the only option is *constant*. Style *constant* simply applies a constant value of respective property to each particle of the assigned type.

The *react* keyword controls whether bonds are created/deleted when particles transition between a fluid and solid state. This option only applies to atom types that have a defined value of *Tfreeze*. When a fluid particle’s temperature drops below *Tfreeze*, bonds of type *btype* are created between nearby solid particles within a distance of *cut*. The particle’s status also swaps to a solid state. When a solid particle’s temperature rises above *Tfreeze*, all bonds of type *btype* are broken and the particle’s status swaps to a fluid state.

2.189.4 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](#).

2.189.5 Restrictions

This fix must be used with an atom style that includes temperature, heatflow, and conductivity such as atom_style rheo/thermal. This fix must be used in conjunction with [fix rheo](#) with the *thermal* setting. The fix group must be set to all. Only one instance of fix rheo/pressure can be defined.

This fix is part of the RHEO package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

2.189.6 Related commands

[fix rheo](#), [pair rheo](#), [compute rheo/property/atom](#), [fix add/heat](#)

2.189.7 Default

none

2.190 fix rheo/viscosity command

2.190.1 Syntax

```
fix ID group-ID rheo/viscosity type1 pstyle1 args1 ... typeN pstyleN argsN
```

- ID, group-ID are documented in [fix](#) command
- rheo/viscosity = style name of this fix command
- one or more types and viscosity styles must be appended
- types = lists of types (see below)
- vstyle = *constant* or *power*

constant args = eta
eta = viscosity

power args = eta, gd0, K, n
eta = viscosity
gd0 = critical strain rate
K = consistency index
n = power-law exponent

2.190.2 Examples

```
fix 1 all rheo/viscosity * constant 1.0
fix 1 all rheo/viscosity 1 constant 1.0 2 power 0.1 5e-4 0.001 0.5
```

2.190.3 Description

Added in version 29Aug2024.

This fix defines a viscosity for RHEO particles. One can define different viscosities for different atom types, but a viscosity must be specified for every atom type.

One first defines the atom *types*. A wild-card asterisk can be used in place of or in conjunction with the *types* argument to set the coefficients for multiple pairs of atom types. This takes the form “*” or “*n” or “m*” or “m*n”. If *N* is 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 *m* to *N* (inclusive). A middle asterisk means all types from *m* to *n* (inclusive).

The *types* definition is followed by the viscosity style, *vstyle*. Two options are available, *constant* and *power*. Style *constant* simply applies a constant value of the viscosity *eta* to each particle of the assigned type. Style *power* is a Hershchel-Bulkley constitutive equation for the stress τ

$$\tau = \left(\frac{\tau_0}{\dot{\gamma}} + K\dot{\gamma}^{n-1} \right) \dot{\gamma}, \tau \geq \tau_0$$

where $\dot{\gamma}$ is the strain rate and τ_0 is the critical yield stress, below which $\dot{\gamma} = 0.0$. To avoid divergences, this expression is regularized by defining a critical strain rate *gd0*. If the local strain rate on a particle falls below this limit, a constant viscosity of *eta* is assigned. This implies a value of

$$\tau_0 = \eta\dot{\gamma}_0 - K\dot{\gamma}_0^N$$

2.190.4 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*.

2.190.5 Restrictions

This fix must be used with an atom style that includes viscosity such as atom_style rheo or rheo/thermal. This fix must be used in conjunction with *fix rheo*. The fix group must be set to all. Only one instance of fix rheo/viscosity can be defined.

This fix is part of the RHEO package. It is only enabled if LAMMPS was built with that package. See the *Build package* page for more info.

2.190.6 Related commands

fix rheo, *pair rheo*, *compute rheo/property/atom*

2.190.7 Default

none

2.191 fix rhok command

2.191.1 Syntax

```
fix ID group-ID rhok nx ny nz K a
```

- ID, group-ID are documented in [fix](#) command
- nx, ny, nz = k-vector of collective density field
- K = spring constant of bias potential
- a = anchor point of bias potential

2.191.2 Examples

```
fix bias all rhok 16 0 0 4.0 16.0
fix 1 all npt temp 0.8 0.8 4.0 z 2.2 2.2 8.0
# output of 4 values from fix rhok: U_bias rho_k_RE rho_k_IM \rho_k\
thermo_style custom step temp pzz lz f_bias f_bias[1] f_bias[2] f_bias[3]
```

2.191.3 Description

The fix applies a force to atoms given by the potential

$$\begin{aligned} U &= \frac{1}{2} K(|\rho_{\vec{k}}| - a)^2 \\ \rho_{\vec{k}} &= \sum_j^N \exp(-i\vec{k} \cdot \vec{r}_j) / \sqrt{N} \\ \vec{k} &= (2\pi n_x / L_x, 2\pi n_y / L_y, 2\pi n_z / L_z) \end{aligned}$$

as described in ([Pedersen](#)).

This field, which biases configurations with long-range order, can be used to study crystal-liquid interfaces and determine melting temperatures ([Pedersen](#)).

An example of using the interface pinning method is located in the *examples/PACKAGES/rhok* directory.

2.191.4 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 calculated by the fix to the global potential energy of the system as part of *thermodynamic output*. The default setting for this fix is *fix_modify energy no*.

This fix computes a global scalar which can be accessed by various *output commands*. The scalar is the potential energy discussed in the preceding paragraph. 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.

This fix is not invoked during *energy minimization*.

2.191.5 Restrictions

This fix is part of the EXTRA-FIX package. It is only enabled if LAMMPS was built with that package. See the *Build package* page for more info.

2.191.6 Related commands

thermo_style

2.191.7 Default

none

(Pedersen) Pedersen, J. Chem. Phys., 139, 104102 (2013).

2.192 fix rigid command

Accelerator Variants: *rigid/omp*

2.193 fix rigid/nve command

Accelerator Variants: *rigid/nve/omp*

2.194 fix rigid/nvt command

Accelerator Variants: *rigid/nvt/omp*

2.195 fix rigid/npt command

Accelerator Variants: *rigid/npt/omp*

2.196 fix rigid/nph command

Accelerator Variants: *rigid/nph/omp*

2.197 fix rigid/small command

Accelerator Variants: *rigid/small/omp*

2.198 fix rigid/nve/small command

2.199 fix rigid/nvt/small command

2.200 fix rigid/npt/small command

2.201 fix rigid/nph/small command

2.201.1 Syntax

`fix ID group-ID style bodystyle args keyword values ...`

- ID, group-ID are documented in [fix](#) command
- style = *rigid* or *rigid/nve* or *rigid/nvt* or *rigid/npt* or *rigid/nph* or *rigid/small* or *rigid/nve/small* or *rigid/nvt/small* or *rigid/npt/small* or *rigid/nph/small*
- bodystyle = *single* or *molecule* or *group*

single args = none

molecule args = none

custom args = *i_propname* or *v_varname*

i_propname = a custom integer vector defined via fix property/atom

v_varname = an atom-style or atomfile-style variable

group args = N groupID1 groupID2 ...

 N = # of groups

 groupID1, groupID2, ... = list of N group IDs

- zero or more keyword/value pairs may be appended

- keyword = *langevin* or *reinit* or *temp* or *mol* or *iso* or *aniso* or *x* or *y* or *z* or *couple* or *tparam* or *pchain* or *dilate* or *force* or *torque* or *infile* or *gravity*

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)
 reinit value = yes or no
 temp values = Tstart Tstop Tdamp
 Tstart,Tstop = desired temperature at start/stop of run (temperature units)
 Tdamp = temperature damping parameter (time units)
 mol value = template-ID
 template-ID = ID of molecule template specified in a separate molecule command
 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 value = 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
 gravity values = gravity-ID
 gravity-ID = ID of fix gravity command to add gravitational forces

2.201.2 Examples

```

fix 1 clump rigid single reinit yes
fix 1 clump rigid/small molecule
fix 1 clump rigid single force 1 off off on langevin 1.0 1.0 1.0 428984
fix 1 polychains rigid/nvt molecule temp 1.0 1.0 5.0 reinit no
fix 1 polychains rigid molecule force 1*5 off off off force 6*10 off off on
fix 1 polychains rigid/small molecule langevin 1.0 1.0 1.0 428984
fix 2 fluid rigid group 3 clump1 clump2 clump3 torque * off off off
fix 1 rods rigid/npt molecule temp 300.0 300.0 100.0 iso 0.5 0.5 10.0
fix 1 particles rigid/npt molecule temp 1.0 1.0 5.0 x 0.5 0.5 1.0 z 0.5 0.5 1.0 couple xz
fix 1 water rigid/nph molecule iso 0.5 0.5 1.0
fix 1 particles rigid/npt/small molecule temp 1.0 1.0 1.0 iso 0.5 0.5 1.0

variable bodyid atom 1.0*gmask(clump1)+2.0*gmask(clump2)+3.0*gmask(clump3)
fix 1 clump rigid custom v_bodyid

variable bodyid atomfile bodies.txt
fix 1 clump rigid custom v_bodyid

```

(continues on next page)

(continued from previous page)

```
fix 0 all property/atom i_bodyid
read_restart data.rigid fix 0 NULL Bodies
fix 1 clump rigid/small custom i_bodyid
```

2.201.3 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. The coordinates, velocities, and orientations of the atoms in each body are then updated so that the body moves and rotates as a single entity. This is implemented by creating internal data structures for each rigid body and performing time integration on these data structures. Positions, velocities, and orientations of the constituent particles are regenerated from the rigid body data structures in every time step. This restricts which operations and fixes can be applied to rigid bodies. See below for a detailed discussion.

Examples of large rigid bodies are a colloidal particle, or portions of a 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 consisting 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, NVT, NPT, or NPH ensemble, as described below.

There are two main variants of this fix, fix rigid and fix rigid/small. The NVE/NVT/NPT/NHT versions belong to one of the two variants, as their style names indicate.

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* styles are 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.

The *rigid/small* styles are typically best for a system with a large number of small rigid bodies. Each body is assigned to the atom closest to the geometrical center of the body. The fix operates using local lists of rigid bodies owned by each processor and information is exchanged and summed via local communication between neighboring processors when ghost atom info is accumulated.

Note

To use the *rigid/small* styles the ghost atom cutoff must be large enough to span the distance between the atom that owns the body and every other atom in the body. This distance value is printed out when the rigid bodies are defined. If the [pair_style](#) cutoff plus neighbor skin does not span this distance, then you should use the [comm_modify cutoff](#) command with a setting epsilon larger than the distance.

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 may accumulate to produce divergent trajectories.

Note

You should not update the atoms in rigid bodies via other time-integration fixes (e.g. [fix nve](#), [fix nvt](#), [fix npt](#), [fix move](#)), or you will have conflicting updates to positions and velocities resulting in unphysical behavior in most cases. When performing a hybrid simulation with some atoms in rigid bodies, and some not, a separate time integration fix like [fix nve](#) or [fix nvt](#) should be used for the non-rigid particles.

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).

Warning

The aggregate properties of each rigid body are calculated at the start of a simulation run and are maintained in internal data structures. The properties include the position and velocity of the center-of-mass of the body, its moments of inertia, and its angular momentum. This is done using the properties of the constituent atoms of the body at that point in time (or see the [infile](#) keyword option). Thereafter, changing these properties of individual atoms in the body will have no effect on a rigid body’s dynamics, unless they effect any computation of per-atom forces or torques. If the keyword [reinit](#) is set to *yes* (the default), the rigid body data structures will be recreated at the beginning of each [run](#) command; if the keyword [reinit](#) is set to *no*, the rigid body data structures will be built only at the very first [run](#) command and maintained for as long as the rigid fix is defined. For example, you might think you could displace the atoms in a body or add a large velocity to each atom in a body to make it move in a desired direction before a second run is performed, using the [set](#) or [displace_atoms](#) or [velocity](#) commands. But these commands will not affect the internal attributes of the body unless [reinit](#) is set to *yes*. With [reinit](#) set to *no* (or using the [infile](#) option, which implies [reinit no](#)) the position and velocity of individual atoms in the body will be reset when time integration starts again.

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.

Note

With the *rigid/small* styles, which require that *bodystyle* be specified as *molecule* or *custom*, you can define a system that has no rigid bodies initially. This is useful when you are using the *mol* keyword in conjunction with another fix that is adding rigid bodies on-the-fly as molecules, such as [fix deposit](#) or [fix pour](#).

For *bodystyle single* the entire fix group of atoms is treated as one rigid body. This option is only allowed for the *rigid* styles.

For bodystyle *molecule*, atoms are grouped into rigid bodies by their respective molecule IDs: each set of atoms in the fix group with the same molecule ID is treated as a different rigid body. This option is allowed for both the *rigid* and *rigid/small* 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.

Bodystyle *custom* is similar to bodystyle *molecule* except that it is more flexible in using other per-atom properties to define the sets of atoms that form rigid bodies. A custom per-atom integer vector defined by the *fix property/atom* command can be used. Or an *atom-style* or *atomfile-style variable* can be used; the floating-point value produced by the variable is rounded to an integer. As with bodystyle *molecule*, each set of atoms in the fix groups with the same integer value is treated as a different rigid body. Since fix property/atom custom vectors and atom-style variables produce values for all atoms, you should be careful to use a fix group that only includes atoms you want to be part of rigid bodies.

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 ensure 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 the *rigid* 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 ensure 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 or 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.

Note

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. If the rigid bodies have strongly overlapping atoms, you may need to turn off these interactions to avoid numerical problems due to large equal/opposite intra-body forces swamping the contribution of small inter-body forces.

For computational efficiency, you should typically define one fix rigid or fix rigid/small command which includes all the desired rigid bodies. LAMMPS 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 LAMMPS) or finite-size particles, such as spheres or ellipsoids or line segments or triangles. See the [atom_style sphere and 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*, *rigid/nve*, *rigid/small*, and *rigid/small/nve* styles perform constant NVE time integration. They are referred to below as the 4 NVE rigid styles. The only difference is that the *rigid* and *rigid/small* styles use an integration technique based on Richardson iterations. The *rigid/nve* and *rigid/small/nve* styles uses the methods described in the paper by [Miller](#), which are thought to provide better energy conservation than an iterative approach.

The *rigid/nvt* and *rigid/nvt/small* styles performs constant NVT integration using a Nose/Hoover thermostat with chains as described originally in ([Hoover](#)) and ([Martyna](#)), which thermostats both the translational and rotational degrees of freedom of the rigid bodies. They are referred to below as the 2 NVT rigid styles. The rigid-body algorithm used by *rigid/nvt* is described in the paper by [Kamberaj](#).

The *rigid/npt*, *rigid/nph*, *rigid/npt/small*, and *rigid/nph/small* styles perform constant NPT or NPH integration using a Nose/Hoover barostat with chains. They are referred to below as the 4 NPT and NPH rigid styles. For the NPT case, the same Nose/Hoover thermostat is also used as with *rigid/nvt* and *rigid/nvt/small*.

The barostat parameters are specified using one or more of the *iso*, *aniso*, *x*, *y*, *z* and *couple* keywords. These keywords give you the ability to specify 3 diagonal components of the external stress tensor, and to couple these components together so that the dimensions they represent are varied together during a constant-pressure simulation. The effects of these keywords are similar to those defined in [fix npt/nph](#)

Note

Currently the *rigid/npt*, *rigid/nph*, *rigid/npt/small*, and *rigid/nph/small* styles do not support triclinic (non-orthogonal) boxes.

The target pressures for each of the 6 components of the stress tensor can be specified independently via the *x*, *y*, *z* keywords, which correspond to the 3 simulation box dimensions. For each component, the external pressure or tensor component at each timestep is a ramped value during the run from *Pstart* to *Pstop*. If a target pressure is specified for a component, then the corresponding box dimension will change during a simulation. For example, if the *y* keyword is used, the *y*-box length will change. A box dimension will not change if that component is not specified, although you have the option to change that dimension via the [fix deform](#) command.

For all barostat keywords, the *Pdamp* parameter operates like the *Tdamp* parameter, determining the time scale on which pressure is relaxed. For example, a value of 10.0 means to relax the pressure in a timespan of (roughly) 10 time units (e.g. *tau* or *fs* or *ps* - see the [units](#) command).

Regardless of what atoms are in the fix group (the only atoms which are time integrated), a global pressure or stress tensor is computed for all atoms. Similarly, when the size of the simulation box is changed, all atoms are re-scaled to new positions, unless the keyword *dilate* is specified with a *dilate-group-ID* for a group that represents a subset of the atoms. This can be useful, for example, to leave the coordinates of atoms in a solid substrate unchanged and controlling the pressure of a surrounding fluid. Another example is a system consisting of rigid bodies and point particles where the barostat is only coupled with the rigid bodies. This option should be used with care, since it can be unphysical to dilate some atoms and not others, because it can introduce large, instantaneous displacements between a pair of atoms (one dilated, one not) that are far from the dilation origin.

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 P_{xx} and P_{zz} components of the stress tensor are coupled. *Xyz* means all 3 diagonal components are coupled. Coupling means two things: the instantaneous stress will be computed as an average of the corresponding diagonal components, and the coupled box dimensions will be changed together in lockstep, meaning coupled dimensions will be dilated or contracted by the same percentage every timestep. The *Pstart*, *Pstop*, *Pdamp* parameters for any coupled dimensions must be identical. *Couple xyz* can be used for a 2d simulation; the *z* dimension is simply ignored.

The *iso* and *aniso* keywords are simply shortcuts that are equivalent to specifying several other keywords together.

The keyword *iso* means couple all 3 diagonal components together when pressure is computed (hydrostatic pressure), and dilate/contract the dimensions together. Using “*iso Pstart Pstop Pdamp*” is the same as specifying these 4 keywords:

```
x Pstart Pstop Pdamp
y Pstart Pstop Pdamp
z Pstart Pstop Pdamp
couple xyz
```

The keyword *aniso* means *x*, *y*, and *z* dimensions are controlled independently using the P_{xx} , P_{yy} , and P_{zz} components of the stress tensor as the driving forces, and the specified scalar external pressure. Using “*aniso Pstart Pstop Pdamp*” is the same as specifying these 4 keywords:

```
x Pstart Pstop Pdamp
y Pstart Pstop Pdamp
z Pstart Pstop Pdamp
couple none
```

The keyword/value option pairs are used in the following ways.

The *reinit* keyword determines, whether the rigid body properties are re-initialized between run commands. With the option *yes* (the default) this is done, with the option *no* this is not done. Turning off the re-initialization can be helpful to protect rigid bodies against unphysical manipulations between runs or when properties cannot be easily re-computed (e.g. when read from a file). When using the *infile* keyword, the *reinit* option is automatically set to *no*.

The *langevin* and *temp* and *tparam* keywords perform thermostatting of the rigid bodies, altering both their translational and rotational degrees of freedom. What is meant by “temperature” of a collection of rigid bodies and how it can be monitored via the fix output is discussed below.

The *langevin* keyword applies a Langevin thermostat to the constant NVE time integration performed by any of the 4 NVE rigid styles: *rigid*, *rigid/nve*, *rigid/small*, *rigid/small/nve*. It cannot be used with the 2 NVT rigid styles: *rigid/nvt*, *rigid/small/nvt*. The desired temperature at each timestep is a ramped value during the run from *Tstart* to *Tstop*. The *Tdamp* parameter is specified in time units and determines how rapidly the temperature is relaxed. For example, a value of 100.0 means to relax the temperature in a timespan of (roughly) 100 time units (τ or fs or ps - see the *units* command). The random # *seed* must be a positive integer.

The way that Langevin thermostatting operates is explained on the [fix langevin](#) doc page. If you wish to simply viscously damp the rotational motion without thermostatting, you can set *Tstart* and *Tstop* to 0.0, which means only the viscous drag term in the Langevin thermostat will be applied. See the discussion on the [fix viscous](#) page for details.

Note

When the *langevin* keyword is used with *fix rigid* versus *fix rigid/small*, different dynamics will result for parallel runs. This is because of the way random numbers are used in the two cases. The dynamics for the two cases should be statistically similar, but will not be identical, even for a single timestep.

The *temp* and *tparam* keywords apply a Nose/Hoover thermostat to the NVT time integration performed by the 2 NVT rigid styles. They cannot be used with the 4 NVE rigid styles. The desired temperature at each timestep is a ramped value during the run from *Tstart* to *Tstop*. The *Tdamp* parameter is specified in time units and determines how rapidly the temperature is relaxed. For example, a value of 100.0 means to relax the temperature in a timespan of (roughly) 100 time units (tau or fs or ps - see the *units* command).

Nose/Hoover chains are used in conjunction with this thermostat. The *tchain* keyword can optionally be used to change the chain settings used. *Tchain* is the number of thermostats in the Nose Hoover chain. This value, along with *Tdamp* can be varied to dampen undesirable oscillations in temperature that can occur in a simulation. As a rule of thumb, increasing the chain length should lead to smaller oscillations. The keyword *pchain* specifies the number of thermostats in the chain thermostating the barostat degrees of freedom.

Note

There are alternate ways to thermostat a system of rigid bodies. You can use *fix langevin* to treat the individual particles in the rigid bodies as effectively immersed in an implicit solvent, e.g. a Brownian dynamics model. For hybrid systems with both rigid bodies and solvent particles, you can thermostat only the solvent particles that surround one or more rigid bodies by appropriate choice of groups in the compute and fix commands for temperature and thermostating. The solvent interactions with the rigid bodies should then effectively thermostat the rigid body temperature as well without use of the Langevin or Nose/Hoover options associated with the fix rigid commands.

The *mol* keyword can only be used with the *rigid/small* styles. It must be used when other commands, such as *fix deposit* or *fix pour*, add rigid bodies on-the-fly during a simulation. You specify a *template-ID* previously defined using the *molecule* command, which reads a file that defines the molecule. You must use the same *template-ID* that the other fix which is adding rigid bodies uses. The coordinates, atom types, atom diameters, center-of-mass, and moments of inertia can be specified in the molecule file. See the *molecule* command for details. The only settings required to be in this file are the coordinates and types of atoms in the molecule, in which case the molecule command calculates the other quantities itself.

Note that these other fixes create new rigid bodies, in addition to those defined initially by this fix via the *bodystyle* setting.

Also note that when using the *mol* keyword, extra restart information about all rigid bodies is written out whenever a restart file is written out. See the NOTE in the next section for details.

The *infile* keyword allows a file of rigid body attributes to be read in from a file, rather than having LAMMPS compute them. There are 5 such attributes: the total mass of the rigid body, its center-of-mass position, its 6 moments of inertia, its center-of-mass velocity, and the 3 image flags of the center-of-mass position. For rigid bodies consisting of point particles or non-overlapping finite-size particles, LAMMPS can compute these values accurately.

However, for rigid bodies consisting of finite-size particles which overlap each other, LAMMPS will ignore the overlaps when computing these 4 attributes, which means the dynamics of the bodies will be incorrect. 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 vxcm vycm vzcm lx ly lz ixcm iycm izcm
ID2 masstotal xcm ycm zcm ixx iyy izz ixy ixz iyz vxcm vycm vzcm lx ly lz ixcm iycm izcm
...
IDN masstotal xcm ycm zcm ixx iyy izz ixy ixz iyz vxcm vycm vzcm lx ly lz ixcm iycm izcm
```

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, LAMMPS 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 principal axes of the rigid body itself. LAMMPS performs the latter calculation internally.

The (vxcm,vycm,vzcm) values are the velocity of the center of mass. The (lx,ly,lz) values are the angular momentum of the body. The (vxcm,vycm,vzcm) and (lx,ly,lz) values can simply be set to 0 if you wish the body to have no initial motion.

The (ixcm,iycm,izcm) values are the image flags of the center of mass of the body. For periodic dimensions, they specify which image of the simulation box the body is considered to be in. An image of 0 means it is inside the box as defined. A value of 2 means add 2 box lengths to get the true value. A value of -1 means subtract 1 box length to get the true value. LAMMPS updates these flags as the rigid bodies cross periodic boundaries during the simulation.

Note

If you use the *infile* or *mol* keywords and write restart files during a simulation, then each time a restart file is written, the fix also writes 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. 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.

If the system has rigid bodies with finite-size overlapping particles and the model uses the *fix gravity* command to apply a gravitational force to the rigid bodies, then the *gravity* keyword should be used in the following manner.

First, the group specified for the *fix gravity* command should not include any atoms in rigid bodies which have overlapping particles. It can be empty (see the *group empty* command) or only contain single particles not in rigid bodies, e.g. background particles.

Second, the *infile* keyword should be used to specify the total mass and other properties of the rigid bodies with overlaps, so that their dynamics will be modeled correctly, as explained above.

Third, the *gravity* keyword should be used with the ID of the *fix gravity* command as its argument. The rigid fixes will access the gravity fix to extract the current direction of the gravity vector at each timestep (which can be static or dynamic). A gravity force will then be applied to each rigid body at its center-of-mass position using its total mass.

If you use a *temperature compute* with a group that includes particles in rigid bodies, the degrees-of-freedom removed by each rigid body are accounted for in the temperature (and pressure) computation, but only if the temperature group includes all the particles in a particular rigid body.

A 3d rigid body has 6 degrees of freedom (3 translational, 3 rotational), except for a collection of point particles lying on a straight line, which has only 5, e.g. a dimer. A 2d rigid body has 3 degrees of freedom (2 translational, 1 rotational).

Note

You may wish to explicitly subtract additional degrees-of-freedom if you use the *force* and *torque* keywords to eliminate certain motions of one or more rigid bodies. LAMMPS does not do this automatically.

The rigid body contribution to the pressure of the system (virial) is also accounted for by this fix.

If your simulation is a hybrid model with a mixture of rigid bodies and non-rigid particles (e.g. solvent) there are several ways these rigid fixes can be used in tandem with [fix nve](#), [fix nvt](#), [fix npt](#), and [fix nph](#).

If you wish to perform NVE dynamics (no thermostating or barostatting), use one of 4 NVE rigid styles to integrate the rigid bodies, and [fix nve](#) to integrate the non-rigid particles.

If you wish to perform NVT dynamics (thermostatting, but no barostatting), you can use one of the 2 NVT rigid styles for the rigid bodies, and any thermostating fix for the non-rigid particles ([fix nvt](#), [fix langevin](#), [fix temp/berendsen](#)). You can also use one of the 4 NVE rigid styles for the rigid bodies and thermostat them using [fix langevin](#) on the group that contains all the particles in the rigid bodies. The net force added by [fix langevin](#) to each rigid body effectively thermostats its translational center-of-mass motion. Not sure how well it does at thermostatting its rotational motion.

If you wish to perform NPT or NPH dynamics (barostatting), you cannot use both [fix npt](#) and the NPT or NPH rigid styles. This is because there can only be one fix which monitors the global pressure and changes the simulation box dimensions. So you have 3 choices:

1. Use one of the 4 NPT or NPH styles for the rigid bodies. Use the *dilate* all option so that it will dilate the positions of the non-rigid particles as well. Use [fix nvt](#) (or any other thermostat) for the non-rigid particles.
2. Use [fix npt](#) for the group of non-rigid particles. Use the *dilate* all option so that it will dilate the center-of-mass positions of the rigid bodies as well. Use one of the 4 NVE or 2 NVT rigid styles for the rigid bodies.
3. Use [fix press/berendsen](#) to compute the pressure and change the box dimensions. Use one of the 4 NVE or 2 NVT rigid styles for the rigid bodies. Use [fix nvt](#) (or any other thermostat) for the non-rigid particles.

In all case, the rigid bodies and non-rigid particles both contribute to the global pressure and the box is scaled the same by any of the barostatting fixes.

You could even use the second and third options for a non-hybrid simulation consisting of only rigid bodies, assuming you give [fix npt](#) an empty group, though it's an odd thing to do. The barostatting fixes ([fix npt](#) and [fix press/berendsen](#)) will monitor the pressure and change the box dimensions, but not time integrate any particles. The integration of the rigid bodies will be performed by fix rigid/nvt.

Styles with a *gpu*, *intel*, *kk*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed on the [Accelerator packages](#) page. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the GPU, INTEL, KOKKOS, OPENMP, and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the [Build package](#) page for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the [-suffix command-line switch](#) when you invoke LAMMPS, or you can use the [suffix](#) command in your input script.

See the [Accelerator packages](#) page for more instructions on how to use the accelerated styles effectively.

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

No information about the 4 NVE rigid styles is written to *binary restart files*. The exception is if the *infile* or *mol* keyword is used, in which case an auxiliary file is written out with rigid body information each time a restart file is written, as explained above for the *infile* keyword. For the 2 NVT rigid styles, the state of the Nose/Hoover thermostat is written to *binary restart files*. Ditto for the 4 NPT and NPH rigid styles, and the state of the Nose/Hoover barostat. See the *read_restart* command for info on how to re-specify a fix in an input script that reads a restart file, so that the operation of the fix continues in an uninterrupted fashion.

The *fix_modify temp* and *press* options are supported by the 4 NPT and NPH rigid styles to change the computes used to calculate the instantaneous pressure tensor. Note that the 2 NVT rigid fixes do not use any external compute to compute instantaneous temperature.

The *fix_modify bodyforces* option is supported by all rigid styles to set whether per-body forces and torques are computed early or late in a timestep, i.e. at the post-force stage or at the final-integrate stage or the timestep, respectively.

The cumulative energy change in the system imposed by the 6 NVT, NPT, NPH rigid fixes, via either thermostating and/or barostatting, is included in the *thermodynamic output* keywords *ecouple* and *econserve*. See the *thermo_style* doc page for details.

The 2 NVE rigid 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.

The 6 NVT, NPT, NPH rigid fixes compute a global scalar which can be accessed by various *output commands*. The scalar is the same cumulative energy change due to these fixes described above. The scalar value calculated by this fix is “extensive”.

The *fix_modify virial* option is supported by these fixes to add the contribution due to the added forces on atoms to both the global pressure and per-atom stress of the system via the *compute pressure* and *compute stress/atom* commands. The former can be accessed by *thermodynamic output*. The default setting for this fix is *fix_modify virial yes*.

All of the *rigid* styles (but not the *rigid/small* styles) compute a global array of values which can be accessed by various *output commands*. Similar information about the bodies defined by the *rigid/small* styles can be accessed via the *compute rigid/local* command.

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.

The center of mass (COM) for each body is similar to unwrapped coordinates written to a dump file. It will always be inside (or slightly outside) the simulation box. The image flags have the same meaning as image flags for atom positions (see the “dump” command). This means you can calculate the unwrapped COM by applying the image flags to the COM, the same as when unwrapped coordinates are written to a dump file.

The force and torque values in the array are not affected by the *force* and *torque* keywords in the fix rigid command; they reflect values before any changes are made by those keywords.

The ordering of the rigid bodies (by row in the array) is as follows. For the *single* keyword there is just one rigid body. For the *molecule* keyword, the bodies are ordered by ascending molecule ID. For the *group* keyword, the list of group IDs determines the ordering of bodies.

The array values calculated by these fixes are “intensive”, meaning they are independent of the number of atoms in the simulation.

No parameter of these fixes can be used with the *start/stop* keywords of the [run](#) command. These fixes are not invoked during *energy minimization*.

2.201.5 Restrictions

These fixes are all part of the RIGID package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

Assigning a temperature via the [velocity create](#) command to a system with *rigid bodies* may not have the desired outcome for two reasons. First, the velocity command can be invoked before the rigid-body fix is invoked or initialized and the number of adjusted degrees of freedom (DOFs) is known. Thus it is not possible to compute the target temperature correctly. Second, the assigned velocities may be partially canceled when constraints are first enforced, leading to a different temperature than desired. A workaround for this is to perform a [run 0](#) command, which ensures all DOFs are accounted for properly, and then rescale the temperature to the desired value before performing a simulation. For example:

```
velocity all create 300.0 12345
run 0                                # temperature may not be 300K
velocity all scale 300.0                # now it should be
```

2.201.6 Related commands

[delete_bonds](#), [neigh_modify exclude](#), [fix shake](#)

2.201.7 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, reinit = yes.

(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).

2.202 fix rigid/meso command

2.202.1 Syntax

```
fix ID group-ID rigid/meso bodystyle args keyword values ...
```

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

- `bodystyle = single or molecule or group`

`single args = none`

`molecule args = none`

`custom args = i_propname or v_varname`

`i_propname = an integer property defined via fix property/atom`

`v_varname = an atom-style or atomfile-style variable`

`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 = `reinit` or `force` or `torque` or `infile`

`reinit = yes or no`

`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`

2.202.2 Examples

```
fix 1 ellipsoid rigid/meso single
fix 1 rods    rigid/meso molecule
fix 1 spheres rigid/meso single force 1 off off on
fix 1 particles rigid/meso molecule force 1*5 off off off force 6*10 off off on
fix 2 spheres rigid/meso group 3 sphere1 sphere2 sphere3 torque * off off off
```

2.202.3 Description

Treat one or more sets of mesoscopic SPH/SDPD particles 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. The coordinates and velocities of the particles in each body are then updated so that the body moves and rotates as a single entity using the methods described in the paper by ([Miller](#)). Density and internal energy of the particles will also be updated. This is implemented by creating internal data structures for each rigid body and performing time integration on these data structures. Positions and velocities of the constituent particles are regenerated from the rigid body data structures in every time step. This restricts which operations and fixes can be applied to rigid bodies. See below for a detailed discussion.

The operation of this fix is exactly like that described by the `fix rigid/nve` command, except that particles' density, internal energy and extrapolated velocity are also updated.

Note

You should not update the particles in rigid bodies via other time-integration fixes (e.g. `fix sph`, `fix sph/stationary`), or you will have conflicting updates to positions and velocities resulting in unphysical behavior in most cases. When performing a hybrid simulation with some atoms in rigid bodies, and some not, a separate time integration fix like `fix sph` should be used for the non-rigid particles.

Note

These fixes are overkill if you simply want to hold a collection of particles stationary or have them move with a constant velocity. To hold particles stationary use [fix sph/stationary](#) instead. If you would like to move particles with a constant velocity use [fix meso/move](#).

Warning

The aggregate properties of each rigid body are calculated at the start of a simulation run and are maintained in internal data structures. The properties include the position and velocity of the center-of-mass of the body, its moments of inertia, and its angular momentum. This is done using the properties of the constituent particles of the body at that point in time (or see the [infile](#) keyword option). Thereafter, changing these properties of individual particles in the body will have no effect on a rigid body's dynamics, unless they effect any computation of per-particle forces or torques. If the keyword [reinit](#) is set to *yes* (the default), the rigid body data structures will be recreated at the beginning of each [run](#) command; if the keyword [reinit](#) is set to *no*, the rigid body data structures will be built only at the very first [run](#) command and maintained for as long as the rigid fix is defined. For example, you might think you could displace the particles in a body or add a large velocity to each particle in a body to make it move in a desired direction before a second run is performed, using the [set](#) or [displace_atoms](#) or [velocity](#) commands. But these commands will not affect the internal attributes of the body unless [reinit](#) is set to *yes*. With [reinit](#) set to *no* (or using the [infile](#) option, which implies [reinit no](#)) the position and velocity of individual particles in the body will be reset when time integration starts again.

Each rigid body must have two or more particles. A particle can belong to at most one rigid body. Which particles are in which bodies can be defined via several options.

For bodystyle *single* the entire fix group of particles is treated as one rigid body.

For bodystyle *molecule*, particles are grouped into rigid bodies by their respective molecule IDs: each set of particles in the fix group with the same molecule ID is treated as a different rigid body. Note that particles with a molecule ID = 0 will be treated as a single rigid body. For a system with solvent (typically this is particles 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 particles you want to be part of rigid bodies.

Bodystyle *custom* is similar to bodystyle *molecule* except that it is more flexible in using other per-atom properties to define the sets of particles that form rigid bodies. An integer vector defined by the [fix property/atom](#) command can be used. Or an [atom-style](#) or [atomfile-style](#) variable can be used; the floating-point value produced by the variable is rounded to an integer. As with bodystyle *molecule*, each set of particles in the fix groups with the same integer value is treated as a different rigid body. Since fix property/atom vectors and atom-style variables produce values for all particles, you should be careful to use a fix group that only includes particles you want to be part of rigid bodies.

For bodystyle *group*, each of the listed groups is treated as a separate rigid body. Only particles that are also in the fix group are included in each rigid body.

Note

To compute the initial center-of-mass position and other properties of each rigid body, the image flags for each particle in the body are used to “unwrap” the particle coordinates. Thus you must ensure that these image flags are consistent so that the unwrapping creates a valid rigid body (one where the particles are close together), particularly if the particles 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 particle 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 particle must be 0 in that dimension, else an error is generated.

By default, each rigid body is acted on by other particles 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 ensure 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 or 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 bodies 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 should typically define one fix rigid/meso command which includes all the desired rigid bodies. LAMMPS will allow multiple rigid/meso fixes to be defined, but it is more expensive.

The keyword/value option pairs are used in the following ways.

The *reinit* keyword determines, whether the rigid body properties are re-initialized between run commands. With the option *yes* (the default) this is done, with the option *no* this is not done. Turning off the re-initialization can be helpful to protect rigid bodies against unphysical manipulations between runs or when properties cannot be easily re-computed (e.g. when read from a file). When using the *infile* keyword, the *reinit* option is automatically set to *no*.

The *infile* keyword allows a file of rigid body attributes to be read in from a file, rather than having LAMMPS compute them. There are 5 such attributes: the total mass of the rigid body, its center-of-mass position, its 6 moments of inertia, its center-of-mass velocity, and the 3 image flags of the center-of-mass position. For rigid bodies consisting of point particles or non-overlapping finite-size particles, LAMMPS can compute these values accurately. However, for rigid bodies consisting of finite-size particles which overlap each other, LAMMPS will ignore the overlaps when computing these 4 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 vxcm vycm vzcm lx ly lz ixcm iycm izcm
ID2 masstotal xcm ycm zcm ixx iyy izz ixy ixz iyz vxcm vycm vzcm lx ly lz ixcm iycm izcm
...
IDN masstotal xcm ycm zcm ixx iyy izz ixy ixz iyz vxcm vycm vzcm lx ly lz ixcm iycm izcm
```

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, LAMMPS 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 principal axes of the rigid body itself. LAMMPS performs the latter calculation internally.

The (vxcm,vcym,vzcm) values are the velocity of the center of mass. The (lx,ly,lz) values are the angular momentum of the body. The (vxcm,vcym,vzcm) and (lx,ly,lz) values can simply be set to 0 if you wish the body to have no initial motion.

The (ixcm,iycm,izcm) values are the image flags of the center of mass of the body. For periodic dimensions, they specify which image of the simulation box the body is considered to be in. An image of 0 means it is inside the box as defined. A value of 2 means add 2 box lengths to get the true value. A value of -1 means subtract 1 box length to get the true value. LAMMPS updates these flags as the rigid bodies cross periodic boundaries during the simulation.

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 writes 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. 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.

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

No information is written to *binary restart files*. If the *infile* keyword is used, an auxiliary file is written out with rigid body information each time a restart file is written, as explained above for the *infile* keyword.

None of the *fix_modify* options are relevant to this fix.

This fix computes a global array of values which can be accessed by various *output commands*.

The number of rows in the array is equal to the number of rigid bodies. The number of columns is 28. Thus for each rigid body, 28 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 components of the 4-vector quaternion representing the orientation of the rigid body, the xyz components of the angular velocity of the body around its COM, the xyz components of the torque acting on the COM, the 3 principal components of the moment of inertia, the xyz components of the angular momentum of the body around its COM, and the xyz image flags of the COM.

The center of mass (COM) for each body is similar to unwrapped coordinates written to a dump file. It will always be inside (or slightly outside) the simulation box. The image flags have the same meaning as image flags for particle positions (see the “dump” command). This means you can calculate the unwrapped COM by applying the image flags to the COM, the same as when unwrapped coordinates are written to a dump file.

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 this fix are “intensive”, meaning they are independent of the number of particles in the simulation.

No parameter of this fix can be used with the *start/stop* keywords of the [run](#) command.

This fix is not invoked during [energy minimization](#).

2.202.5 Restrictions

This fix is part of the DPD-SMOOTH package and also depends on the RIGID package. It is only enabled if LAMMPS was built with both packages. See the [Build package](#) page for more info.

This fix requires that atoms store density and internal energy as defined by the [atom_style sph](#) command.

All particles in the group must be mesoscopic SPH/SDPD particles.

Changed in version 29Aug2024.

This fix is incompatible with deformation controls that remap velocity, for instance the *remap v* option of [fix deform](#).

2.202.6 Related commands

[fix meso/move](#), [fix rigid](#), [neigh_modify exclude](#)

2.202.7 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 reinit = yes.

(Miller) Miller, Eleftheriou, Pattnaik, Ndirango, and Newns, J Chem Phys, 116, 8649 (2002).

2.203 fix rx command

Accelerator Variants: *rx/kk*

2.203.1 Syntax

```
fix ID group-ID rx file localTemp matrix solver minSteps ...
```

- ID, group-ID are documented in [fix](#) command
- rx = style name of this fix command
- file = filename containing the reaction kinetic equations and Arrhenius parameters
- localTemp = *none,lucy* = no local temperature averaging or local temperature defined through Lucy weighting function
- matrix = *sparse, dense* format for the stoichiometric matrix

- solver = *lammps_rk4,rkf45* = rk4 is an explicit fourth order Runge-Kutta method; rkf45 is an adaptive fourth-order Runge-Kutta-Fehlberg method
- minSteps = # of steps for rk4 solver or minimum # of steps for rkf45 (rk4 or rkf45)
- maxSteps = maximum number of steps for the rkf45 solver (rkf45 only)
- relTol = relative tolerance for the rkf45 solver (rkf45 only)
- absTol = absolute tolerance for the rkf45 solver (rkf45 only)
- diag = Diagnostics frequency for the rkf45 solver (optional, rkf45 only)

2.203.2 Examples

```
fix 1 all rx kinetics.rx none dense lammps_rk4
fix 1 all rx kinetics.rx none sparse lammps_rk4 1
fix 1 all rx kinetics.rx lucy sparse lammps_rk4 10
fix 1 all rx kinetics.rx none dense rkf45 1 100 1e-6 1e-8
fix 1 all rx kinetics.rx none dense rkf45 1 100 1e-6 1e-8 -1
```

2.203.3 Description

Fix *rx* solves the reaction kinetic ODEs for a given reaction set that is defined within the file associated with this command.

For a general reaction such that



the reaction rate equation is defined to be of the form

$$r = k(T)[A]^{v_A} [B]^{v_B}$$

In the current implementation, the exponents are defined to be equal to the stoichiometric coefficients. A given reaction set consisting of *n* reaction equations will contain a total of *m* species. A set of *m* ordinary differential equations (ODEs) that describe the change in concentration of a given species as a function of time are then constructed based on the *n* reaction rate equations.

The ODE systems are solved over the full DPD timestep *dt* using either a fourth order Runge-Kutta *rk4* method with a fixed step-size *h*, specified by the *lammps_rk4* keyword, or a fourth order Runge-Kutta-Fehlberg (rkf45) method with an adaptive step-size for *h*. The number of ODE steps per DPD timestep for the rk4 method is optionally specified immediately after the rk4 keyword. The ODE step-size is set as *dt/num_steps*. Smaller step-sizes tend to yield more accurate results but there is no control on the error. For error control, use the rkf45 ODE solver.

The rkf45 method adjusts the step-size so that the local truncation error is held within the specified absolute and relative tolerances. The initial step-size *h0* can be specified by the user or estimated internally. It is recommended that the user specify *h0* since this will generally reduce the number of ODE integration steps required. *h0* is defined as *dt/min_steps* if *min_steps* >= 1. If *min_steps* == 0, *h0* is estimated such that an explicit Euler method would likely produce an acceptable solution. This is generally overly conservative for the fourth-order method and users are advised to specify *h0* as some fraction of the DPD timestep. For small DPD timesteps, only one step may be necessary depending upon the tolerances. Note that more than *min_steps* ODE steps may be taken depending upon the ODE stiffness but no more than *max_steps* will be taken. If *max_steps* is reached, an error warning is printed and the simulation is stopped.

After each ODE step, the solution error *e* is tested and weighted using the *absTol* and *relTol* values. The error vector is weighted as *e* / (*relTol* * $|u|$ + *absTol*) where *u* is the solution vector. If the norm of the error is <= 1, the solution

is accepted, h is increased by a proportional amount, and the next ODE step is begun. Otherwise, h is shrunk and the ODE step is repeated.

Run-time diagnostics are available for the rkf45 ODE solver. The frequency (in timesteps) that diagnostics are reported is controlled by the last (optional) 12th argument. A negative frequency means that diagnostics are reported once at the end of each run. A positive value N means that the diagnostics are reported once per N timesteps.

The diagnostics report the average # of integrator steps and RHS function evaluations and run-time per ODE as well as the average/RMS/min/max per process. If the reporting frequency is 1, the RMS/min/max per ODE are also reported. The per ODE statistics can be used to adjust the tolerance and min/max step parameters. The statistics per MPI process can be useful to examine any load imbalance caused by the adaptive ODE solver. (Some DPD particles can take longer to solve than others. This can lead to an imbalance across the MPI processes.)

The filename specifies a file that contains the entire set of reaction kinetic equations and corresponding Arrhenius parameters. The format of this file is described below.

There is no restriction on the total number or reaction equations that are specified. The species names are arbitrary string names that are associated with the species concentrations. Each species in a given reaction must be preceded by its stoichiometric coefficient. The only delimiters that are recognized between the species are either a + or = character. The = character corresponds to an irreversible reaction. After specifying the reaction, the reaction rate constant is determined through the temperature dependent Arrhenius equation:

$$k = AT^n e^{-\frac{E_a}{k_B T}}$$

where A is the Arrhenius factor in time units or concentration/time units, n is the unitless exponent of the temperature dependence, and E_a is the activation energy in energy units. The temperature dependence can be removed by specifying the exponent as zero.

The internal temperature of the coarse-grained particles can be used in constructing the reaction rate constants at every DPD timestep by specifying the keyword *none*. Alternatively, the keyword *lucy* can be specified to compute a local-average particle internal temperature for use in the reaction rate constant expressions. The local-average particle internal temperature is defined as:

$$\theta_i^{-1} = \frac{\sum_{j=1} \omega_{Lucy}(r_{ij}) \theta_j^{-1}}{\sum_{j=1} \omega_{Lucy}(r_{ij})}$$

where the Lucy function is expressed as:

$$\omega_{Lucy}(r_{ij}) = \left(1 + \frac{3r_{ij}}{r_c}\right) \left(1 - \frac{r_{ij}}{r_c}\right)^3$$

The self-particle interaction is included in the above equation.

The stoichiometric coefficients for the reaction mechanism are stored in either a sparse or dense matrix format. The dense matrix should only be used for small reaction mechanisms. The sparse matrix should be used when there are many reactions (e.g., more than 5). This allows the number of reactions and species to grow while keeping the computational cost tractable. The matrix format can be specified as using either the *sparse* or *dense* keywords. If all stoichiometric coefficients for a reaction are small integers (whole numbers ≤ 3), a fast exponential function is used. This can save significant computational time so users are encouraged to use integer coefficients where possible.

The format of a tabulated file is as follows (without the parenthesized comments):

# Rxn equations and parameters	(one or more comment or blank lines)
1.0 hcn + 1.0 no2 = 1.0 no + 0.5 n2 + 0.5 h2 + 1.0 co 2.49E+01 0.0 1.34	(rxn equation, A, n, Ea)
1.0 hcn + 1.0 no = 1.0 co + 1.0 n2 + 0.5 h2 2.16E+00 0.0 1.52	
...	
1.0 no + 1.0 co = 0.5 n2 + 1.0 co2 1.66E+06 0.0 0.69	

A section begins with a non-blank line whose first character is not a “#”; blank lines or lines starting with “#” can be used as comments between sections.

Following a blank line, the next N lines list the N reaction equations. Each species within the reaction equation is specified through its stoichiometric coefficient and a species tag. Reactant species are specified on the left-hand side of the equation and product species are specified on the right-hand side of the equation. After specifying the reactant and product species, the final three arguments of each line represent the Arrhenius parameter *A*, the temperature exponent *n*, and the activation energy *Ea*.

Note that the species tags that are defined in the reaction equations are used by the [fix eos/table/rx](#) command to define the thermodynamic properties of each species. Furthermore, the number of species molecules (i.e., concentration) can be specified either with the [set](#) command using the “d_” prefix or by reading directly the concentrations from a data file. For the latter case, the [read_data](#) command with the fix keyword should be specified, where the fix-ID will be the “fix rx`ID with a <SPECIES>`_ suffix, e.g.

```
fix foo all rx reaction.file ... read_data data.dpd fix foo_SPECIES NULL Species
```

Styles with a *gpu*, *intel*, *kk*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed on the [Accelerator packages](#) page. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the GPU, INTEL, KOKKOS, OPENMP, and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the [Build package](#) page for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the [-suffix command-line switch](#) when you invoke LAMMPS, or you can use the [suffix](#) command in your input script.

See the [Accelerator packages](#) page for more instructions on how to use the accelerated styles effectively.

2.203.4 Restrictions

This command is part of the DPD-REACT package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

This command also requires use of the [atom_style dpd](#) command.

This command can only be used with a constant energy or constant enthalpy DPD simulation.

2.203.5 Related commands

fix eos/table/rx, fix shardlow, pair dpd/fdt/energy

2.203.6 Default

none

2.204 fix saed/vtk command

2.204.1 Syntax

```
fix ID group-ID saed/vtk Nevery Nrepeat Nfreak c_ID attribute args ... keyword args ...
```

- ID, group-ID are documented in [fix](#) command
- saed/vtk = 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
- c_ID = saed compute ID

keyword = file or ave or start or file or overwrite:l

ave args = one or running or window M

one = output a new average value every Nfreq steps

running = output cumulative average of all previous Nfreq steps

window M = output average of M most recent Nfreq steps

start args = Nstart

Nstart = start averaging on this timestep

file arg = filename

filename = name of file to output time averages to

2.204.2 Examples

```
compute 1 all saed 0.0251 Al O Kmax 1.70 Zone 0 0 1 dR_Ewald 0.01 c 0.5 0.5 0.5
compute 2 all saed 0.0251 Ni Kmax 1.70 Zone 0 0 0 c 0.05 0.05 0.05 manual echo
```

```
fix 1 all saed/vtk 1 1 1 c_1 file Al2O3_001.saed
fix 2 all saed/vtk 1 1 1 c_2 file Ni_000.saed
```

2.204.3 Description

Time average computed intensities from [compute saed](#) and write output to a file in the third generation vtk image data format for visualization directly in parallelized visualization software packages like ParaView and VisIt. Note that if no time averaging is done, this command can be used as a convenient way to simply output diffraction intensities at a single snapshot.

To produce output in the image data vtk format ghost data is added outside the K_{max} range assigned in the compute saed. The ghost data is assigned a value of -1 and can be removed setting a minimum isovolume of 0 within the visualization software. SAED images can be created by visualizing a spherical slice of the data that is centered at $R_{Ewald}*[h\ k\ l]/\text{norm}([h\ k\ l])$, where $R_{Ewald}=1/\lambda$.

The group specified within this command is ignored. However, note that specified values may represent calculations performed by saed computes which store their own “group” definitions.

Fix saed/vtk is designed to work only with [compute saed](#) values, e.g.

```
compute 3 top saed 0.0251 Al O
fix saed/vtk 1 1 1 c _3 file Al2O3_001.saed
```

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. *Nrepeat*Nevery* can not exceed *Nfreq*.

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.

The output for fix ave/time/saed is a file written with the third generation vtk image data formatting. The filename assigned by the *file* keyword is appended with *_N.vtk* where *N* is an index (0,1,2...) to account for multiple diffraction intensity outputs.

By default the header contains the following information (with example data):

```
# vtk DataFile Version 3.0 c_SAED
Image data set
ASCII
DATASET STRUCTURED_POINTS
DIMENSIONS 337 219 209
ASPECT_RATIO 0.00507953 0.00785161 0.00821458
ORIGIN -0.853361 -0.855826 -0.854316
POINT_DATA 15424827
SCALARS intensity float
LOOKUP_TABLE default
...data
```

In this example, kspace is sampled across a 337 x 219 x 209 point mesh where the mesh spacing is approximately 0.005, 0.007, and 0.008 inv(length) units in the k1, k2, and k3 directions, respectively. The data is shifted by -0.85, -0.85, -0.85 inv(length) units so that the origin will lie at 0, 0, 0. Here, 15,424,827 kspace points are sampled in total.

Additional optional keywords also affect the operation of this fix.

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 cumulative sense before being output. Each output value is thus the average of the value produced on that timestep with all preceding values. This running average begins when the fix is defined; it can only be restarted by deleting the fix via the *unfix* command, or by re-defining the fix by re-specifying it.

If the *ave* setting is *window*, then the values produced on timesteps that are multiples of *Nfreq* are summed and averaged within a moving “window” of time, so that the last M values are used to produce the output. E.g. if M = 3 and Nfreq = 1000, then the output on step 10000 will be the average of the individual values on steps 8000,9000,10000. Outputs on early steps will average over less than M values if they are not available.

The *start* keyword specifies what timestep averaging will begin on. The default is step 0. Often 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 *file* keyword allows a filename to be specified. Every *Nfreq* steps, the vector of saed intensity data is written to a new file using the third generation vtk format. The base of each file is assigned by the *file* keyword and this string is appended with _N.vtk where N is an index (0,1,2...) to account for situations with multiple diffraction intensity outputs.

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

This fix is part of the DIFFRACTION package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

No information about this fix is written to *binary restart files*. 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*.

2.204.5 Restrictions

The attributes for fix_saed_vtk must match the values assigned in the associated *compute_saed* command.

2.204.6 Related commands

compute_saed

2.204.7 Default

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

(Coleman) Coleman, Spearot, Capolungo, MSMSE, 21, 055020 (2013).

2.205 fix setforce command

Accelerator Variants: *setforce/kk*

2.206 fix setforce/spin command

2.206.1 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

2.206.2 Examples

```
fix freeze indenter setforce 0.0 0.0 0.0
fix 2 edge setforce NULL 0.0 0.0
fix 1 edge setforce/spin 0.0 0.0 0.0
fix 2 edge setforce NULL 0.0 v_oscillate
```

2.206.3 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.

Style *spin* suffix sets the components of the magnetic precession vectors instead of the mechanical forces. This also erases all previously computed magnetic precession vectors on the atom, though additional magnetic fixes could add new forces.

This command can be used to freeze the magnetic moment of certain atoms in the simulation by zeroing their precession vector.

All options defined above remain valid, they just apply to the magnetic precession vectors instead of the forces.

Styles with a *gpu*, *intel*, *kk*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed on the [Accelerator packages](#) page. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the GPU, INTEL, KOKKOS, OPENMP, and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the [Build package](#) page for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the *-suffix command-line switch* when you invoke LAMMPS, or you can use the *suffix* command in your input script.

See the [Accelerator packages](#) page for more instructions on how to use the accelerated styles effectively.

Note

The *region* keyword is supported by Kokkos, but a Kokkos-enabled region must be used. See the *region region* command for more information.

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

No information about this fix is written to *binary restart files*.

The *fix_modify respa* option is supported by this fix. This allows to set at which level of the *r-RESPA* integrator the fix is setting the forces to the desired values; on all other levels, the force is set to 0.0 for the atoms in the fix group, so that setforce values are not counted multiple times. Default is to override forces at the outermost level.

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.

2.206.5 Restrictions

Fix *setforce/spin* is part of the SPIN package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

2.206.6 Related commands

fix addforce, *fix aveforce*

2.206.7 Default

none

2.207 fix sgcmc command

2.207.1 Syntax

```
fix ID group-ID sgcmc every_nsteps swap_fraction temperature deltamu ...
```

- ID, group-ID are documented in [fix](#) command
- sgcmc = style name of this fix command
- every_nsteps = number of MD steps between MC cycles
- swap_fraction = fraction of a full MC cycle carried out at each call (a value of 1.0 will perform as many trial moves as there are atoms)
- temperature = temperature that enters Boltzmann factor in Metropolis criterion (usually the same as MD temperature)
- deltamu = N - I chemical potential differences $\mu_1 - \mu_2, \dots, \mu_1 - \mu_N$ (N is the number of atom types)
- Zero or more keyword/value pairs may be appended to fix definition line:

keyword = variance or randseed or window_moves or window_size

variance kappa conc1 [conc2] ... [concN]

kappa = variance constraint parameter

c_2, c_3,..., c_N = N-1 target concentration fractions

randseed N

N = seed for pseudo random number generator

window_moves N

N = number of times sampling window is moved during one MC cycle

window_size frac

frac = size of sampling window (must be between 0.5 and 1.0)

2.207.2 Examples

```
fix mc all sgcmc 50 0.1 400.0 -0.55
fix vc all sgcmc 20 0.2 700.0 -0.7 randseed 324234 variance 2000.0 0.05
fix 2 all sgcmc 20 0.1 700.0 -0.7 window_moves 20
```

2.207.3 Description

Added in version 22Dec2022.

This command allows to carry out parallel hybrid molecular dynamics/Monte Carlo (MD/MC) simulations using the algorithms described in ([Sadigh1](#)). Simulations can be carried out in either the semi-grand canonical (SGC) or variance constrained semi-grand canonical (VC-SGC) ensemble ([Sadigh2](#)). Only atom type swaps are performed by the SGCMC fix. Relaxations are accounted for by the molecular dynamics integration steps.

This fix can be used with standard multi-element EAM potentials ([pair styles eam/alloy or eam/fs](#))

The SGCMC fix can handle Finnis/Sinclair type EAM potentials where $\rho(r)$ is atom-type specific, such that different elements can contribute differently to the total electron density at an atomic site depending on the identity of the element at that atomic site.

If this fix is applied, the regular MD simulation will be interrupted in defined intervals to carry out a fraction of a Monte Carlo (MC) cycle. The interval is set using the parameter *every_nsteps* which determines how many MD integrator steps are taken between subsequent calls to the MC routine.

It is possible to carry out pure lattice MC simulations by setting *every_nsteps* to 1 and not defining an integration fix such as NVE, NPT etc. In that case, the particles will not move and only the MC routine will be called to perform atom type swaps.

The parameter *swap_fraction* determines how many MC trial steps are carried out every time the MC routine is entered. It is measured in units of full MC cycles where one full cycle, *swap_fraction*=1, corresponds to as many MC trial steps as there are atoms.

The parameter *temperature* specifies the temperature that is used to evaluate the Metropolis acceptance criterion. While it usually should be set to the same value as the MD temperature there are cases when it can be useful to use two different values for at least part of the simulation, e.g., to speed up equilibration at low temperatures.

The parameter *deltamu* is used to set the chemical potential differences in the SGC MC algorithm (see Eq. 16 in [Sadigh1](#)). The $N-1$ differences are defined as $\mu_1 - \mu_2, \dots, \mu_1 - \mu_N$, where N is the number of atom types.

The variance-constrained SGC MC algorithm is activated if the keyword *variance* is used. In that case the fix parameter *deltamu* determines the effective average constraint in the parallel VC-SGC MC algorithm (parameter $\delta\mu_0$ in Eq. (20) of [Sadigh1](#)). The parameter *kappa* specifies the variance constraint (see Eqs. (20-21) in [Sadigh1](#)). The parameter *conc* sets the $N-1$ target atomic concentration fractions (parameter c_0 in Eqs. (20-21) of [Sadigh1](#)) $0 \leq c_2, \dots, c_N \leq 1$, with $c_1 = 1 - \sum_{i=2}^N c_i$. When the simulation includes N atom types (elements), $N-1$ concentration values must be specified.

There are several technical parameters that can be set via optional flags.

randseed is expected to be a positive integer number and is used to initialize the random number generator on each processor.

window_size controls the size of the sampling window in a parallel MC simulation. The size has to lie between 0.5 and 1.0. Normally, this parameter should be left unspecified which instructs the code to choose the optimal window size automatically (see Sect. III.B and Figure 6 in *Sadigh1* for details).

The number of times the window is moved during a MC cycle is set using the parameter *window_moves* (see Sect. III.B in *Sadigh1* for details).

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

No information about this fix is written to restart files.

The MC routine keeps track of the global concentration(s) as well as the number of accepted and rejected trial swaps during each MC step. These values are provided by the sgcmc fix in the form of a global vector that can be accessed by various *output commands* components of the vector represent the following quantities:

- 1 = The absolute number of accepted trial swaps during the last MC step
- 2 = The absolute number of rejected trial swaps during the last MC step
- 3 = Current global concentration c_1 (= number of atoms of type 1 / total number of atoms)
- 4 = Current global concentration c_2 (= number of atoms of type 2 / total number of atoms)
- ...
- N+2 = Current global concentration c_N (= number of atoms of type N / total number of atoms)

The vector values calculated by this fix are “intensive”.

2.207.5 Restrictions

This fix is part of the MC package. It is only enabled if LAMMPS was built with that package. See the *Build package* page for more info.

This fix style requires an *atom style* with per atom type masses.

At present the fix provides optimized subroutines for EAM type potentials (see above) that calculate potential energy changes due to *local* atom type swaps very efficiently. Other potentials are supported by using the generic potential functions. This, however, will lead to exceedingly slow simulations since it implies that the energy of the *entire* system is recomputed at each MC trial step. If other potentials are to be used it is strongly recommended to modify and optimize the existing generic potential functions for this purpose. Also, the generic energy calculation can not be used for parallel execution i.e. it only works with a single MPI process.

2.207.6 Default

The optional parameters default to the following values:

- *randseed* = 324234
- *window_moves* = 8
- *window_size* = automatic

(Sadigh1) B. Sadigh, P. Erhart, A. Stukowski, A. Caro, E. Martinez, and L. Zepeda-Ruiz, Phys. Rev. B **85**, 184203 (2012)

(Sadigh2) B. Sadigh and P. Erhart, Phys. Rev. B **86**, 134204 (2012)

2.208 fix shake command

Accelerator Variants: *shake/kk*

2.209 fix rattle command

2.209.1 Syntax

```
fix ID group-ID style tol iter N constraint values ... keyword value ...
```

- ID, group-ID are documented in [fix](#) command
- style = shake or rattle = style name of this fix command
- tol = accuracy tolerance of SHAKE solution
- iter = max # of iterations in each SHAKE solution
- N = print SHAKE statistics every this many timesteps (0 = never)
- one or more constraint/value pairs are appended
- constraint = *b* or *a* or *t* or *m*
 - b* values = one or more bond types (may use type labels)
 - a* values = one or more angle types (may use type labels)
 - t* values = one or more atom types (may use type labels)
 - m* value = one or more mass values
- zero or more keyword/value pairs may be appended
- keyword = *mol* or *kbond*
 - mol* value = template-ID
 - template-ID = ID of molecule template specified in a separate [molecule](#) command
 - kbond* value = force constant
 - force constant = force constant used to apply a restraint force when used during minimization

2.209.2 Examples

```
fix 1 sub shake 0.0001 20 10 b 4 19 a 3 5 2
fix 1 sub shake 0.0001 20 10 t 5 6 m 1.0 a 31
fix 1 sub shake 0.0001 20 10 t 5 6 m 1.0 a 31 mol myMol
fix 1 sub rattle 0.0001 20 10 t 5 6 m 1.0 a 31
fix 1 sub rattle 0.0001 20 10 t 5 6 m 1.0 a 31 mol myMol
```

2.209.3 Description

Apply bond and angle constraints to specified bonds and angles in the simulation by either the SHAKE or RATTLE algorithms. This typically enables a longer timestep. The SHAKE or RATTLE constraint algorithms, however, can *only* be applied during molecular dynamics runs.

Changed in version 15Sep2022.

These fixes may now also be used during minimization. In that case the constraints are *approximated* by strong harmonic restraints.

SHAKE vs RATTLE:

The SHAKE algorithm was invented for schemes such as standard Verlet timestepping, where only the coordinates are integrated and the velocities are approximated as finite differences to the trajectories ([Ryckaert et al. \(1977\)](#)). If the velocities are integrated explicitly, as with velocity Verlet which is what LAMMPS uses as an integration method, a second set of constraining forces is required in order to eliminate velocity components along the bonds ([Andersen \(1983\)](#)).

In order to formulate individual constraints for SHAKE and RATTLE, focus on a single molecule whose bonds are constrained. Let \mathbf{R}_i and \mathbf{V}_i be the position and velocity of atom i at time n , for $i = 1, \dots, N$, where N is the number of sites of our reference molecule. The distance vector between sites i and j is given by

$$\mathbf{r}_{ij}^{n+1} = \mathbf{r}_j^n - \mathbf{r}_i^n$$

The constraints can then be formulated as

$$\begin{aligned}\mathbf{r}_{ij}^{n+1} \cdot \mathbf{r}_{ij}^{n+1} &= d_{ij}^2 \quad \text{and} \\ \mathbf{v}_{ij}^{n+1} \cdot \mathbf{r}_{ij}^{n+1} &= 0\end{aligned}$$

The SHAKE algorithm satisfies the first condition, i.e. the sites at time $n+1$ will have the desired separations D_{ij} immediately after the coordinates are integrated. If we also enforce the second condition, the velocity components along the bonds will vanish. RATTLE satisfies both conditions. As implemented in LAMMPS, `fix rattle` uses `fix shake` for satisfying the coordinate constraints. Therefore the settings and optional keywords are the same for both fixes, and all the information below about SHAKE is also relevant for RATTLE.

SHAKE:

Each timestep the specified bonds and angles are reset to their equilibrium lengths and angular values via the SHAKE algorithm ([Ryckaert et al. \(1977\)](#)). This is done by applying an additional constraint force so that the new positions preserve the desired atom separations. The equations for the additional force are solved via an iterative method that typically converges to an accurate solution in a few iterations. The desired tolerance (e.g. $1.0e-4 = 1$ part in 10000) and maximum # of iterations are specified as arguments. Setting the `N` argument will print statistics to the screen and log file about regarding the lengths of bonds and angles that are being constrained. Small delta values mean SHAKE is doing a good job.

In LAMMPS, only small clusters of atoms can be constrained. This is so the constraint calculation for a cluster can be performed by a single processor, to enable good parallel performance. A cluster is defined as a central atom connected to others in the cluster by constrained bonds. LAMMPS allows for the following kinds of clusters to be constrained: one central atom bonded to 1 or 2 or 3 atoms, or one central atom bonded to 2 others and the angle between the three atoms also constrained. This means water molecules or CH₂ or CH₃ groups may be constrained, but not all the C-C backbone bonds of a long polymer chain.

The `b` constraint lists bond types that will be constrained. The `t` constraint lists atom types. All bonds connected to an atom of the specified type will be constrained. The `m` constraint lists atom masses. All bonds connected to atoms of the specified masses will be constrained (within a fudge factor of MASSDELTA specified in `src/RIGID/fix_shake.cpp`). The `a` constraint lists angle types. If both bonds in the angle are constrained then the angle will also be constrained if its type is in the list.

Changed in version 29Aug2024.

The types may be given as type labels *only* if there is no atom, bond, or angle type label named *b*, *a*, *t*, or *m* defined in the simulation. If that is the case, type labels cannot be used as constraint type index with these two fixes, because the type labels would be incorrectly treated as a new type of constraint instead. Thus, LAMMPS will print a warning and type label handling is disabled and numeric types must be used.

For all constraints, a particular bond is only constrained if *both* atoms in the bond are in the group specified with the SHAKE fix.

The degrees-of-freedom removed by SHAKE bonds and angles are accounted for in temperature and pressure computations. Similarly, the SHAKE contribution to the pressure of the system (virial) is also accounted for.

Note

This command works by using the current forces on atoms to calculate an additional constraint force which when added will leave the atoms in positions that satisfy the SHAKE constraints (e.g. bond length) after the next time integration step. If you define fixes (e.g. [fix efield](#)) that add additional force to the atoms after [fix shake](#) operates, then this fix will not take them into account and the time integration will typically not satisfy the SHAKE constraints. The solution for this is to make sure that [fix shake](#) is defined in your input script after any other fixes which add or change forces (to atoms that [fix shake](#) operates on).

The *mol* keyword should be used when other commands, such as [fix deposit](#) or [fix pour](#), add molecules on-the-fly during a simulation, and you wish to constrain the new molecules via SHAKE. You specify a *template-ID* previously defined using the [molecule](#) command, which reads a file that defines the molecule. You must use the same *template-ID* that the command adding molecules uses. The coordinates, atom types, special bond restrictions, and SHAKE info can be specified in the molecule file. See the [molecule](#) command for details. The only settings required to be in this file (by this command) are the SHAKE info of atoms in the molecule.

The *kbond* keyword sets the restraint force constant when [fix shake](#) or [fix rattle](#) are used during minimization. In that case the constraint algorithms are *not* applied and restraint forces are used instead to maintain the geometries similar to the constraints. How well the geometries are maintained and how quickly a minimization converges, depends largely on the force constant *kbond*: larger values will reduce the deviation from the desired geometry, but can also lead to slower convergence of the minimization or lead to instabilities depending on the minimization algorithm requiring to reduce the value of [timestep](#). Even though the restraints will not preserve the bond lengths and angles as closely as the constraints during the MD, they are generally close enough so that the constraints will be fulfilled to the desired accuracy within a few MD steps following the minimization. The default value for *kbond* depends on the [units](#) setting and is $1.0\text{e}6 \text{k_B}$.

Styles with a *gpu*, *intel*, *kk*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed on the [Accelerator packages](#) page. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the GPU, INTEL, KOKKOS, OPENMP, and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the [Build package](#) page for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the [-suffix command-line switch](#) when you invoke LAMMPS, or you can use the [suffix](#) command in your input script.

See the [Accelerator packages](#) page for more instructions on how to use the accelerated styles effectively.

RATTLE:

The velocity constraints lead to a linear system of equations which can be solved analytically. The implementation of the algorithm in LAMMPS closely follows ([Andersen \(1983\)](#)).

Note

The `fix rattle` command modifies forces and velocities and thus should be defined after all other integration fixes in your input script. If you define other fixes that modify velocities or forces after `fix rattle` operates, then `fix rattle` will not take them into account and the overall time integration will typically not satisfy the RATTLE constraints. You can check whether the constraints work correctly by setting the value of RATTLE_DEBUG in `src/RIGID/fix_rattle.cpp` to 1 and recompiling LAMMPS.

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

No information about these fixes is written to *binary restart files*.

Both fix `shake` and fix `rattle` behave differently during a minimization in comparison to a molecular dynamics run:

- When used during a minimization, the SHAKE or RATTLE constraint algorithms themselves are **not** applied. Instead the constraints are replaced by harmonic restraint forces. The energy and virial contributions due to the restraint forces are tallied into global and per-atom accumulators. The total restraint energy is also accessible as a global scalar property of the fix.
- During molecular dynamics runs, however, the fixes do apply the requested SHAKE or RATTLE constraint algorithms.

The `fix_modify virial` option is supported by these fixes to add the contribution due to the added constraint forces on atoms to both the global pressure and per-atom stress of the system via the `compute pressure` and `compute stress/atom` commands. The former can be accessed by *thermodynamic output*.

The default setting for this fix is `fix_modify virial yes`. No global or per-atom quantities are stored by these fixes for access by various `output commands` during an MD run. No parameter of these fixes can be used with the `start/stop` keywords of the `run` command.

2.209.5 Restrictions

These fixes are part of the RIGID package. They are only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

For computational efficiency, there can only be one shake or rattle fix defined in a simulation.

If you use a tolerance that is too large or a max-iteration count that is too small, the constraints will not be enforced very strongly, which can lead to poor energy conservation. You can test for this in your system by running a constant NVE simulation with a particular set of SHAKE parameters and monitoring the energy versus time.

SHAKE or RATTLE should not be used to constrain an angle at 180 degrees (e.g. a linear CO₂ molecule). This causes a divergence when solving the constraint equations numerically. You can use `fix rigid` or `fix rigid/small` instead to make a linear molecule rigid.

When used during minimization choosing a too large value of the `kbond` can make minimization very inefficient and also cause stability problems with some minimization algorithms. Sometimes those can be avoided by reducing the `timestep`.

2.209.6 Related commands

fix rigid, fix ehex, fix nve/manifold/rattle

2.209.7 Default

$\text{kbond} = 1.0\text{e}9 * \text{k_B}$

(**Ryckaert**) J.-P. Ryckaert, G. Ciccotti and H. J. C. Berendsen, J of Comp Phys, 23, 327-341 (1977).

(**Andersen**) H. Andersen, J of Comp Phys, 52, 24-34 (1983).

2.210 fix shardlow command

Accelerator Variants: *shardlow/kk*

2.210.1 Syntax

`fix ID group-ID shardlow`

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

2.210.2 Examples

`fix 1 all shardlow`

2.210.3 Description

Specifies that the Shardlow splitting algorithm (SSA) is to be used to integrate the DPD equations of motion. The SSA splits the integration into a stochastic and deterministic integration step. The fix *shardlow* performs the stochastic integration step and must be used in conjunction with a deterministic integrator (e.g. [fix nve](#) or [fix nph](#)). The stochastic integration of the dissipative and random forces is performed prior to the deterministic integration of the conservative force. Further details regarding the method are provided in ([Lisal](#)) and ([Larentzos1](#)).

The fix *shardlow* must be used with the [pair_style dpd/fdt](#) or [pair_style dpd/fdt/energy](#) command to properly initialize the fluctuation-dissipation theorem parameter(s) sigma (and kappa, if necessary).

Note that numerous variants of DPD can be specified by choosing an appropriate combination of the integrator and [pair_style dpd/fdt](#) command. DPD under isothermal conditions can be specified by using `fix shardlow`, `fix nve` and `pair_style dpd/fdt`. DPD under isoenergetic conditions can be specified by using `fix shardlow`, `fix nve` and `pair_style dpd/fdt/energy`. DPD under isobaric conditions can be specified by using `fix shardlow`, `fix nph` and `pair_style dpd/fdt`. DPD under isoenthalpic conditions can be specified by using `fix shardlow`, `fix nph` and `pair_style dpd/fdt/energy`. Examples of each DPD variant are provided in the examples/PACKAGES/dpd-react directory.

Styles with a *gpu*, *intel*, *kk*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed on the [Accelerator packages](#)

page. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the GPU, INTEL, KOKKOS, OPENMP, and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the [Build package](#) page for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the [-suffix command-line switch](#) when you invoke LAMMPS, or you can use the [suffix](#) command in your input script.

See the [Accelerator packages](#) page for more instructions on how to use the accelerated styles effectively.

2.210.4 Restrictions

This command is part of the DPD-REACT package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

This fix is currently limited to orthogonal simulation cell geometries.

This fix must be used with an additional fix that specifies time integration, e.g. [fix nve](#) or [fix nph](#).

The Shardlow splitting algorithm requires the sizes of the subdomain lengths to be larger than twice the cutoff+skin. Generally, the domain decomposition is dependent on the number of processors requested.

2.210.5 Related commands

[pair_style dpd/fdt](#), [fix eos/cv](#)

2.210.6 Default

none

(Lisal) M. Lisal, J.K. Brennan, J. Bonet Avalos, *J. Chem. Phys.*, 135, 204105 (2011).

(Larentzos1) J.P. Larentzos, J.K. Brennan, J.D. Moore, M. Lisal and W.D. Mattson, *Comput. Phys. Commun.*, 185, 1987-1998 (2014).

(Larentzos2) J.P. Larentzos, J.K. Brennan, J.D. Moore, and W.D. Mattson, ARL-TR-6863, U.S. Army Research Laboratory, Aberdeen Proving Ground, MD (2014).

2.211 fix smd command

2.211.1 Syntax

```
fix ID group-ID smd type values keyword values
```

- ID, group-ID are documented in [fix](#) command
- smd = style name of this fix command
- mode = *cvel* or *cfor* to select constant velocity or constant force SMD

cvel values = K vel
 K = spring constant (force/distance units)
 vel = velocity of pulling (distance/time units)
 cfor values = force
 force = pulling force (force units)

- keyword = *tether* or *couple*

tether values = x y z R0
 x,y,z = point to which spring is tethered
 R0 = distance of end of spring from tether point (distance units)
 couple values = group-ID2 x y z R0
 group-ID2 = 2nd group to couple to fix group with a spring
 x,y,z = direction of spring, automatically computed with 'auto'
 R0 = distance of end of spring (distance units)

2.211.2 Examples

```
fix pull cterm smd cvel 20.0 -0.00005 tether NULL NULL 100.0 0.0
fix pull cterm smd cvel 20.0 -0.0001 tether 25.0 25 25.0 0.0
fix stretch cterm smd cvel 20.0 0.0001 couple nterm auto auto auto 0.0
fix pull cterm smd cfor 5.0 tether 25.0 25.0 25.0 0.0
```

2.211.3 Description

This fix implements several options of steered MD (SMD) as reviewed in ([Izrailev](#)), which allows to induce conformational changes in systems and to compute the potential of mean force (PMF) along the assumed reaction coordinate ([Park](#)) based on Jarzynski's equality ([Jarzynski](#)). This fix borrows a lot from [fix spring](#) and [fix setforce](#).

You can apply a moving spring force to a group of atoms (*tether* style) or between two groups of atoms (*couple* style). The spring can then be used in either constant velocity (*cvel*) mode or in constant force (*cfor*) mode to induce transitions in your systems. When running in *tether* style, you may need some way to fix some other part of the system (e.g. via [fix spring/self](#))

The *tether* style attaches a spring between a point at a distance of R0 away from a fixed point x,y,z and the center of mass of the fix group of atoms. A restoring force of magnitude K (R - R0) Mi / M is applied to each atom in the group where K is the spring constant, Mi is the mass of the atom, and M is the total mass of all atoms in the group. Note that K thus represents the total force on the group of atoms, not a per-atom force.

In *cvel* mode the distance R is incremented or decremented monotonously according to the pulling (or pushing) velocity. In *cfor* mode a constant force is added and the actual distance in direction of the spring is recorded.

The *couple* style links two groups of atoms together. The first group is the fix group; the second is specified by group-ID2. The groups are coupled together by a spring that is at equilibrium when the two groups are displaced by a vector in direction x,y,z with respect to each other and at a distance R0 from that displacement. Note that x,y,z only provides a direction and will be internally normalized. But since it represents the *absolute* displacement of group-ID2 relative to the fix group, (1,1,0) is a different spring than (-1,-1,0). For each vector component, the displacement can be described with the *auto* parameter. In this case the direction is re-computed in every step, which can be useful for steering a local process where the whole object undergoes some other change. When the relative positions and distance between the two groups are not in equilibrium, the same spring force described above is applied to atoms in each of the two groups.

For both the *tether* and *couple* styles, any of the x,y,z values can be specified as NULL which means do not include that dimension in the distance calculation or force application.

For constant velocity pulling (*cvel* mode), the running integral over the pulling force in direction of the spring is recorded and can then later be used to compute the potential of mean force (PMF) by averaging over multiple independent trajectories along the same pulling path.

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

The fix stores the direction of the spring, current pulling target distance and the running PMF to *binary restart files*. See the [read_restart](#) command for info on how to re-specify a fix in an input script that reads a restart file, so that the operation of the fix continues in an uninterrupted fashion.

The [fix_modify virial](#) option is supported by this fix to add the contribution due to the added forces on atoms to both the global pressure and per-atom stress of the system via the [compute pressure](#) and [compute stress/atom](#) commands. The former can be accessed by [thermodynamic output](#). The default setting for this fix is *fix_modify virial no*.

The [fix_modify respa](#) option is supported by this fix. This allows to set at which level of the *r-RESPA* integrator the fix is adding its forces. Default is the outermost level.

This fix computes a vector list of 7 quantities, which can be accessed by various [output commands](#). The quantities in the vector are in this order: the x-, y-, and z-component of the pulling force, the total force in direction of the pull, the equilibrium distance of the spring, the distance between the two reference points, and finally the accumulated PMF (the sum of pulling forces times displacement).

The force is the total force on the group of atoms by the spring. In the case of the *couple* style, it is the force on the fix group (group-ID) or the negative of the force on the second group (group-ID2). The vector values calculated by this fix are “extensive”.

No parameter of this fix can be used with the *start/stop* keywords of the [run](#) command. This fix is not invoked during [energy minimization](#).

2.211.5 Restrictions

This fix is part of the EXTRA-FIX package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

2.211.6 Related commands

fix drag, fix spring, fix spring/self, fix spring/rg, fix colvars, fix plumed

2.211.7 Default

none

(Izrailev) Izrailev, Stepaniants, Isralewitz, Kosztin, Lu, Molnar, Wriggers, Schulten. Computational Molecular Dynamics: Challenges, Methods, Ideas, volume 4 of Lecture Notes in Computational Science and Engineering, pp. 39-65. Springer-Verlag, Berlin, 1998.

(Park) Park, Schulten, J. Chem. Phys. 120 (13), 5946 (2004)

(Jarzynski) Jarzynski, Phys. Rev. Lett. 78, 2690 (1997)

2.212 fix smd/adjust_dt command

2.212.1 Syntax

```
fix ID group-ID smd/adjust_dt arg
```

- ID, group-ID are documented in [fix](#) command
- smd/adjust_dt = style name of this fix command
- arg = *s_fact*
s_fact = safety factor

2.212.2 Examples

```
fix 1 all smd/adjust_dt 0.1
```

2.212.3 Description

The fix calculates a new stable time increment for use with the SMD time integrators.

The stable time increment is based on multiple conditions. For the SPH pair styles, a CFL criterion (Courant, Friedrichs & Lewy, 1928) is evaluated, which determines the speed of sound cannot propagate further than a typical spacing between particles within a single time step to ensure no information is lost. For the contact pair styles, a linear analysis of the pair potential determines a stable maximum time step.

This fix inquires the minimum stable time increment across all particles contained in the group for which this fix is defined. An additional safety factor *s_fact* is applied to the time increment.

See [this PDF guide](#) to use Smooth Mach Dynamics in LAMMPS.

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

Currently, no part of MACHDYN supports restarting nor minimization.

2.212.5 Restrictions

This fix is part of the MACHDYN package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

2.212.6 Related commands

[smd/tlsph_dt](#)

2.212.7 Default

none

2.213 fix smd/integrate_tlsph command

2.213.1 Syntax

```
fix ID group-ID smd/integrate_tlsph keyword values
```

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

limit_velocity value = max_vel
max_vel = maximum allowed velocity

2.213.2 Examples

```
fix 1 all smd/integrate_tlsph  
fix 1 all smd/integrate_tlsph limit_velocity 1000
```

2.213.3 Description

The fix performs explicit time integration for particles which interact according with the Total-Lagrangian SPH pair style.

See [this PDF guide](#) to using Smooth Mach Dynamics in LAMMPS.

The *limit_velocity* keyword will control the velocity, scaling the norm of the velocity vector to max_vel in case it exceeds this velocity limit.

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

Currently, no part of MACHDYN supports restarting nor minimization. This fix has no outputs.

2.213.5 Restrictions

This fix is part of the MACHDYN package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

Changed in version 29Aug2024.

This fix is incompatible with deformation controls that remap velocity, for instance the *remap v* option of [fix deform](#).

2.213.6 Related commands

smd/integrate_ulsph

2.213.7 Default

none

2.214 fix **smd/integrate_ulsph** command

2.214.1 Syntax

```
fix ID group-ID smd/integrate_ulsph keyword
```

- ID, group-ID are documented in [fix](#) command
- smd/integrate_ulsph = style name of this fix command
- zero or more keyword/value pairs may be appended
- keyword = `adjust_radius` or `limit_velocity`

adjust_radius values = adjust_radius_factor min_nn max_nn

adjust_radius_factor = factor which scale the smooth/kernel radius
min_nn = minimum number of neighbors
max_nn = maximum number of neighbors

limit_velocity values = max_velocity

max_velocity = maximum allowed velocity.

2.214.2 Examples

```
fix 1 all smd/integrate_ulsph adjust_radius 1.02 25 50
fix 1 all smd/integrate_ulsph limit_velocity 1000
```

2.214.3 Description

The fix performs explicit time integration for particles which interact with the updated Lagrangian SPH pair style.

See [this PDF guide](#) to using Smooth Mach Dynamics in LAMMPS.

The `adjust_radius` keyword activates dynamic adjustment of the per-particle SPH smoothing kernel radius such that the number of neighbors per particles remains within the interval `min_nn` to `max_nn`. The parameter `adjust_radius_factor` determines the amount of adjustment per timestep. Typical values are `adjust_radius_factor = 1.02`, `min_nn = 15`, and `max_nn = 20`.

The `limit_velocity` keyword will control the velocity, scaling the norm of the velocity vector to `max_vel` in case it exceeds this velocity limit.

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

Currently, no part of MACHDYN supports restarting nor minimization. This fix has no outputs.

2.214.5 Restrictions

This fix is part of the MACHDYN package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

Changed in version 29Aug2024.

This fix is incompatible with deformation controls that remap velocity, for instance the *remap v* option of [fix deform](#).

2.214.6 Related commands

none

2.214.7 Default

none

2.215 fix smd/move_tri_surf command

2.215.1 Syntax

```
fix ID group-ID smd/move_tri_surf keyword
```

- ID, group-ID are documented in [fix](#) command
- smd/move_tri_surf keyword = style name of this fix command
- keyword = *LINEAR or *WIGGLE or *ROTATE

*LINEAR args = Vx Vy Vz

Vx,Vy,Vz = components of velocity vector (velocity units), any component can be specified as $\textcolor{red}{_}$ NULL

*WIGGLE args = Vx Vy Vz max_travel

vx,vy,vz = components of velocity vector (velocity units), any component can be specified as $\textcolor{red}{_}$ NULL
max_travel = wiggle amplitude

*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)

2.215.2 Examples

```
fix 1 tool smd/move_tri_surf *LINEAR 20 20 10
fix 2 tool smd/move_tri_surf *WIGGLE 20 20 10
fix 2 tool smd/move_tri_surf *ROTATE 0 0 0 5 2 1
```

2.215.3 Description

This fix applies only to rigid surfaces read from .STL files via fix [smd/wall_surface](#). It updates position and velocity for the particles in the group each timestep without regard to forces on the particles. The rigid surfaces can thus be moved along simple trajectories during the simulation.

The **LINEAR* style moves particles with the specified constant velocity vector $\mathbf{V} = (V_x, V_y, V_z)$. This style also sets the velocity of each particle to $\mathbf{V} = (V_x, V_y, V_z)$.

The **WIGGLE* style moves particles in an oscillatory fashion. Particles are moved along (v_x, v_y, v_z) with constant velocity until a displacement of *max_travel* is reached. Then, the velocity vector is reversed. This process is repeated.

The **ROTATE* style rotates particles around a rotation axis $\mathbf{R} = (R_x, R_y, R_z)$ that goes through a point $\mathbf{P} = (P_x, P_y, P_z)$. The period of the rotation is also specified. This style also sets the velocity of each particle to $(\omega \times \mathbf{R}_{\perp})$ where ω is its angular velocity around the rotation axis and \mathbf{R}_{\perp} is a perpendicular vector from the rotation axis to the particle.

See this PDF [guide](#) to using Smooth Mach Dynamics in LAMMPS.

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

Currently, no part of MACHDYN supports restarting nor minimization. This fix has no outputs.

2.215.5 Restrictions

This fix is part of the MACHDYN package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

2.215.6 Related commands

[smd/triangle_mesh_vertices](#), [smd/wall_surface](#)

2.215.7 Default

none

2.216 fix smd/setvel command

2.216.1 Syntax

```
fix ID group-ID smd/setvel vx vy vz keyword value ...
```

- ID, group-ID are documented in [fix](#) command
- smd/setvel = style name of this fix command
- vx,vy,vz = velocity component values
- any of vx,vy,vz 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 particles must be in to have their velocities set

2.216.2 Examples

```
fix top_velocity top_group smd/setvel 1.0 0.0 0.0
```

2.216.3 Description

Set each component of velocity on each particle in the group to the specified values vx,vy,vz, regardless of the forces acting on the particle. This command can be used to impose velocity boundary conditions.

Any of the vx,vy,vz values can be specified as NULL which means do not alter the velocity component in that dimension.

This fix is intended to be used together with a time integration fix.

Any of the 3 quantities defining the velocity components can be specified as an equal-style or atom-style *variable*, namely vx, vy, vz. 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 velocity 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 velocity field with optional time-dependence as well.

If the *region* keyword is used, the particle must also be in the specified geometric [region](#) in order to have its velocity set by this command.

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

Currently, no part of MACHDYN supports restarting nor minimization. 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. 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.

2.216.5 Restrictions

This fix is part of the MACHDYN package. It is only enabled if LAMMPS was built with that package. See the *Build package* page for more info.

2.216.6 Related commands

none

2.216.7 Default

none

2.217 fix smd/wall_surface command

2.217.1 Syntax

```
fix ID group-ID smd/wall_surface arg type mol-ID
```

- ID, group-ID are documented in *fix* command
- smd/wall_surface = style name of this fix command
- arg = *file*
file = file name of a triangular mesh in stl format
- type = particle type to be given to the new particles created by this fix
- mol-ID = molecule-ID to be given to the new particles created by this fix (must be ≥ 65535)

2.217.2 Examples

```
fix stl_surf all smd/wall_surface tool.stl 2 65535
```

2.217.3 Description

This fix creates reads a triangulated surface from a file in .STL format. For each triangle, a new particle is created which stores the barycenter of the triangle and the vertex positions. The radius of the new particle is that of the minimum circle which encompasses the triangle vertices.

The triangulated surface can be used as a complex rigid wall via the *smd/tri_surface* pair style. It is possible to move the triangulated surface via the *smd/move_tri_surf* fix style.

Immediately after a .STL file has been read, the simulation needs to be run for 0 timesteps in order to properly register the new particles in the system. See the “funnel_flow” example in the MACHDYN examples directory.

See [this PDF guide](#) to use Smooth Mach Dynamics in LAMMPS.

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

Currently, no part of MACHDYN supports restarting nor minimization. This fix has no outputs.

2.217.5 Restrictions

This fix is part of the MACHDYN package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

The molecule ID given to the particles created by this fix have to be equal to or larger than 65535.

Within each .STL file, only a single triangulated object must be present, even though the STL format allows for the possibility of multiple objects in one file.

2.217.6 Related commands

smd/triangle_mesh_vertices, *smd/move_tri_surf*, *smd/tri_surface*

2.217.7 Default

none

2.218 fix sph command

2.218.1 Syntax

```
fix ID group-ID sph
```

- ID, group-ID are documented in *fix* command
- sph = style name of this fix command

2.218.2 Examples

```
fix 1 all sph
```

2.218.3 Description

Perform time integration to update position, velocity, internal energy and local density for atoms in the group each timestep. This fix is needed to time-integrate SPH systems where particles carry internal variables such as internal energy. SPH stands for Smoothed Particle Hydrodynamics.

See this [PDF guide](#) to using SPH in LAMMPS.

2.218.4 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](#).

2.218.5 Restrictions

This fix is part of the SPH package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

2.218.6 Related commands

[fix sph/stationary](#)

2.218.7 Default

none

2.219 fix sph/stationary command

2.219.1 Syntax

```
fix ID group-ID sph/stationary
```

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

2.219.2 Examples

```
fix 1 boundary sph/stationary
```

2.219.3 Description

Perform time integration to update internal energy and local density, but not position or velocity for atoms in the group each timestep. This fix is needed for SPH simulations to correctly time-integrate fixed boundary particles which constrain a fluid to a given region in space. SPH stands for Smoothed Particle Hydrodynamics.

See this [PDF guide](#) to using SPH in LAMMPS.

2.219.4 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](#).

2.219.5 Restrictions

This fix is part of the SPH package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

2.219.6 Related commands

fix sph

2.219.7 Default

none

2.220 fix spring command

2.220.1 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)

2.220.2 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
```

2.220.3 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 does not 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 $K(R - R_0) M_i / M$ is applied to each atom in the group where K is the spring constant, M_i is the mass of the atom, and M is the total mass of all atoms in the group. Note that K thus represents the spring constant for the total force on the group of atoms, not for a spring applied to each atom.

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).

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.

2.220.4 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 global potential energy of the system as part of *thermodynamic output*. The default setting for this fix is *fix_modify energy no*.

The *fix_modify respa* option is supported by this fix. This allows to set at which level of the *r-RESPA* integrator the fix is adding its forces. Default is the outermost level.

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 second group (group-ID2). The fourth 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.

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.

2.220.5 Restrictions

none

2.220.6 Related commands

fix drag, fix spring/self, fix spring/rg, fix smd

2.220.7 Default

none

2.221 fix spring/chunk command

2.221.1 Syntax

```
fix ID group-ID spring/chunk K chunkID comID
```

- ID, group-ID are documented in *fix* command
- spring/chunk = style name of this fix command

- K = spring constant for each chunk (force/distance units)
- chunkID = ID of *compute chunk/atom* command
- comID = ID of *compute com/chunk* command

2.221.2 Examples

```
fix restrain all spring/chunk 100 chunkID comID
```

2.221.3 Description

Apply a spring force to the center-of-mass (COM) of chunks of atoms as defined by the *compute chunk/atom* command. Chunks can be molecules or spatial bins or other groupings of atoms. This is a way of tethering each chunk to its initial COM coordinates.

The *chunkID* is the ID of a compute chunk/atom command defined in the input script. It is used to define the chunks. The *comID* is the ID of a compute com/chunk command defined in the input script. It is used to compute the COMs of each chunk.

At the beginning of the first *run* or *minimize* command after this fix is defined, the initial COM of each chunk is calculated and stored as R_{0m} , where M is the chunk number. Thereafter, at every timestep (or minimization iteration), the current COM of each chunk is calculated as R_m . A restoring force of magnitude $K (R_m - R_{0m}) M_i / M_m$ is applied to each atom in each chunk where K is the specified spring constant, M_i is the mass of the atom, and M_m is the total mass of all atoms in the chunk. Note that K thus represents the spring constant for the total force on each chunk of atoms, not for a spring applied to each atom.

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

This fix writes the locations of the initial per-chunk center of mass coordinates to *binary restart files*. See the *read_restart* command for info on how to re-specify a fix in an input script that reads a restart file, so that the fix continues in an uninterrupted fashion. Since this fix depends on an instance of *compute chunk/atom* it will check when reading the restart if the chunk still exists and will define the same number of chunks. The restart data is only applied when the number of chunks matches. Otherwise the center of mass coordinates are recomputed.

The *fix_modify energy* option is supported by this fix to add the energy stored in all the springs to the global potential energy of the system as part of *thermodynamic output*. The default setting for this fix is *fix_modify energy no*.

The *fix_modify respa* option is supported by this fix. This allows to set at which level of the *r-RESPA* integrator the fix is adding its forces. Default is the outermost level.

This fix computes a global scalar which can be accessed by various *output commands*. The scalar is the energy of all the springs, i.e. $0.5 * K * r^2$ per-spring.

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.

Note

If you want the spring energies 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.

2.221.5 Restrictions

none

2.221.6 Related commands

fix spring, fix spring/self, fix spring/rg

2.221.7 Default

none

2.222 fix spring/rg command

2.222.1 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

2.222.2 Examples

```
fix 1 protein spring/rg 5.0 10.0
fix 2 micelle spring/rg 5.0 NULL
```

2.222.3 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.

$$\begin{aligned} R_G^2 &= \frac{1}{M} \sum_i^N m_i \left(x_i - \frac{1}{M} \sum_j^N m_j x_j \right)^2 \\ E &= K (R_G - R_{G0})^2 \\ F_i &= 2K \frac{m_i}{M} \left(1 - \frac{R_{G0}}{R_G} \right) \left(x_i - \frac{1}{M} \sum_j^N m_j x_j \right) \end{aligned}$$

The $(x_i - \text{center-of-mass})$ 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 R_{G0} 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.

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

This fix writes the currently used reference RG (R_{G0}) to *binary restart files*. See the [read_restart](#) command for info on how to re-specify a fix in an input script that reads a restart file, so that the fix continues in an uninterrupted fashion.

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 reference radius of gyration R_{G0} used by the fix. energy change due to this fix. 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*.

The [fix_modify respa](#) option is supported by this fix. This allows to set at which level of the *r-RESPA* integrator the fix is adding its forces. Default is the outermost level.

2.222.5 Restrictions

This fix is part of the EXTRA-FIX package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

2.222.6 Related commands

[fix spring](#), [fix spring/self](#) [fix drag](#), [fix smd](#)

2.222.7 Default

none

2.223 fix spring/self command

2.223.1 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)

2.223.2 Examples

```
fix tether boundary-atoms spring/self 10.0
fix zrest move spring/self 10.0 z
```

2.223.3 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 $-K_r$, 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 initial 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.

2.223.4 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 global potential energy of the system as part of *thermodynamic output*. The default setting for this fix is `fix_modify energy no`.

The `fix_modify respa` option is supported by this fix. This allows to set at which level of the *r-RESPA* integrator the fix is adding its forces. Default is the outermost level.

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.

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.

Styles with a *gpu*, *intel*, *kk*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed on the [Accelerator packages](#) page. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the GPU, INTEL, KOKKOS, OPENMP, and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the [Build package](#) page for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the `-suffix command-line switch` when you invoke LAMMPS, or you can use the `suffix` command in your input script.

See the [Accelerator packages](#) page for more instructions on how to use the accelerated styles effectively.

2.223.5 Restrictions

none

2.223.6 Related commands

fix drag, fix spring, fix smd, fix spring/rg

2.223.7 Default

none

2.224 fix srd command

2.224.1 Syntax

```
fix ID group-ID srd N groupbig-ID Tsrd hgrid seed keyword value ...
```

- ID, group-ID are documented in [fix](#) command
- srd = style name of this fix command
- N = reset SRD particle velocities every this many timesteps
- groupbig-ID = ID of group of large particles that SRDs interact with
- Tsrd = temperature of SRD particles (temperature units)
- hgrid = grid spacing for SRD grouping (distance units)
- seed = random # seed (positive integer)
- zero or more keyword/value pairs may be appended
- keyword = *lamda* or *collision* or *overlap* or *inside* or *exact* or *radius* or *bounce* or *search* or *cubic* or *shift* or *tstat* or *rescale*

lamda value = mean free path of SRD particles (distance units)

collision value = noslip or slip = collision model

overlap value = yes or no = whether big particles may overlap

inside value = error or warn or ignore = how SRD particles which end up inside a big particle are *→treated*

exact value = yes or no

radius value = rfactor = scale collision radius by this factor

bounce value = Nbounce = max # of collisions an SRD particle can undergo in one timestep

search value = sgrid = grid spacing for collision partner searching (distance units)

cubic values = style tolerance

style = error or warn

tolerance = fractional difference allowed ($0 \leq tol \leq 1$)

shift values = flag shiftseed

flag = yes or no or possible = SRD bin shifting for better statistics

yes = perform bin shifting each time SRD velocities are rescaled

no = no shifting

possible = shift depending on mean free path and bin size

shiftseed = random # seed (positive integer)
 tstat value = yes or no = thermostat SRD particles or not
 rescale value = yes or no or rotate or collide = rescaling of SRD velocities
 yes = rescale during velocity rotation and collisions
 no = no rescaling
 rotate = rescale during velocity rotation, but not collisions
 collide = rescale during collisions, but not velocity rotation

2.224.2 Examples

```
fix 1 srd srd 10 big 1.0 0.25 482984
fix 1 srd srd 10 big 0.5 0.25 482984 collision slip search 0.5
```

2.224.3 Description

Treat a group of particles as stochastic rotation dynamics (SRD) particles that serve as a background solvent when interacting with big (colloidal) particles in groupbig-ID. The SRD formalism is described in ([Hecht](#)). The same methodology is also called multi-particle collision dynamics (MPCD) in the literature.

The key idea behind using SRD particles as a cheap coarse-grained solvent is that SRD particles do not interact with each other, but only with the solute particles, which in LAMMPS can be spheroids, ellipsoids, or line segments, or triangles, or rigid bodies containing multiple spheroids or ellipsoids or line segments or triangles. The collision and rotation properties of the model imbue the SRD particles with fluid-like properties, including an effective viscosity. Thus simulations with large solute particles can be run more quickly, to measure solute properties like diffusivity and viscosity in a background fluid. The usual LAMMPS fixes for such simulations, such as [fix deform](#), [fix viscosity](#), and [fix nvt/sllod](#), can be used in conjunction with the SRD model.

These 3 papers give more details on how the SRD model is implemented in LAMMPS. ([Petersen](#)) describes pure SRD fluid systems. ([Bolintineanu1](#)) describes models where pure SRD fluids interact with boundary walls. ([Bolintineanu2](#)) describes mixture models where large colloidal particles are solvated by an SRD fluid. See the examples/srd directory for sample input scripts.

This fix does two things:

1. It advects the SRD particles, performing collisions between SRD and big particles or walls every timestep, imparting force and torque to the big particles. Collisions also change the position and velocity of SRD particles.
2. It resets the velocity distribution of SRD particles via random rotations every N timesteps.

SRD particles have a mass, temperature, characteristic timestep dt_{SRD} , and mean free path between collisions (λ). The fundamental equation relating these 4 quantities is

$$\lambda = dt_{SRD} \sqrt{\frac{k_B T_{SRD}}{m}}$$

The mass m of SRD particles is set by the [mass](#) command elsewhere in the input script. The SRD timestep dt_{SRD} is N times the step dt defined by the [timestep](#) command. Big particles move in the normal way via a time integration [fix](#) with a short timestep dt . SRD particles advect with a large timestep $dt_{SRD} \geq dt$.

If the *lambda* keyword is not specified, the SRD temperature T_{SRD} is used in the above formula to compute λ . If the *lambda* keyword is specified, then the *Tsrd* setting is ignored and the above equation is used to compute the SRD temperature.

The characteristic length scale for the SRD fluid is set by *hgrid* which is used to bin SRD particles for purposes of resetting their velocities. Normally *hgrid* is set to be 1/4 of the big particle diameter or smaller, to adequately resolve fluid properties around the big particles.

λ cannot be smaller than $0.6 * hgrid$, else an error is generated (unless the *shift* keyword is used, see below). The velocities of SRD particles are bounded by V_{max} , which is set so that an SRD particle will not advect further than $D_{max} = 4\lambda$ in dt_{SRD} . This means that roughly speaking, D_{max} should not be larger than a big particle diameter, else SRDs may pass through big particles without colliding. A warning is generated if this is the case.

Collisions between SRD particles and big particles or walls are modeled as a lightweight SRD point particle hitting a heavy big particle of given diameter or a wall at a point on its surface and bouncing off with a new velocity. The collision changes the momentum of the SRD particle. It imparts a force and torque to the big particle. It imparts a force to a wall. Static or moving SRD walls are setup via the [fix wall/srd](#) command. For the remainder of this doc page, a collision of an SRD particle with a wall can be viewed as a collision with a big particle of infinite radius and mass.

The *collision* keyword sets the style of collisions. The *slip* style means that the tangential component of the SRD particle momentum is preserved. Thus a force is imparted to a big particle, but no torque. The normal component of the new SRD velocity is sampled from a Gaussian distribution at temperature T_{srdf} .

For the *noslip* style, both the normal and tangential components of the new SRD velocity are sampled from a Gaussian distribution at temperature T_{srdf} . Additionally, a new tangential direction for the SRD velocity is chosen randomly. This collision style imparts torque to a big particle. Thus a time integrator [fix](#) that rotates the big particles appropriately should be used.

The *overlap* keyword should be set to *yes* if two (or more) big particles can ever overlap. This depends on the pair potential interaction used for big-big interactions, or could be the case if multiple big particles are held together as rigid bodies via the [fix rigid](#) command. If the *overlap* keyword is *no* and big particles do in fact overlap, then SRD/big collisions can generate an error if an SRD ends up inside two (or more) big particles at once. How this error is treated is determined by the *inside* keyword. Running with *overlap* set to *no* allows for faster collision checking, so it should only be set to *yes* if needed.

The *inside* keyword determines how a collision is treated if the computation determines that the timestep started with the SRD particle already inside a big particle. If the setting is *error* then this generates an error message and LAMMPS stops. If the setting is *warn* then this generates a warning message and the code continues. If the setting is *ignore* then no message is generated. One of the output quantities logged by the fix (see below) tallies the number of such events, so it can be monitored. Note that once an SRD particle is inside a big particle, it may remain there for several steps until it drifts outside the big particle.

The *exact* keyword determines how accurately collisions are computed. A setting of *yes* computes the time and position of each collision as SRD and big particles move together. A setting of *no* estimates the position of each collision based on the end-of-timestep positions of the SRD and big particle. If *overlap* is set to *yes*, the setting of the *exact* keyword is ignored since time-accurate collisions are needed.

The *radius* keyword scales the effective size of big particles. If big particles will overlap as they undergo dynamics, then this keyword can be used to scale down their effective collision radius by an amount *rfactor*, so that SRD particle will only collide with one big particle at a time. For example, in a Lennard-Jones system at a temperature of 1.0 (in reduced LJ units), the minimum separation between two big particles is as small as about 0.88 sigma. Thus an *rfactor* value of 0.85 should prevent dual collisions.

The *bounce* keyword can be used to limit the maximum number of collisions an SRD particle undergoes in a single timestep as it bounces between nearby big particles. Note that if the limit is reached, the SRD can be left inside a big particle. A setting of 0 is the same as no limit.

There are 2 kinds of bins created and maintained when running an SRD simulation. The first are “SRD bins” which are used to bin SRD particles and reset their velocities, as discussed above. The second are “search bins” which are used to identify SRD/big particle collisions.

The *search* keyword can be used to choose a search bin size for identifying SRD/big particle collisions. The default is to use the *hgrid* parameter for SRD bins as the search bin size. Choosing a smaller or large value may be more efficient, depending on the problem. But, in a statistical sense, it should not change the simulation results.

The *cubic* keyword can be used to generate an error or warning when the bin size chosen by LAMMPS creates SRD bins that are non-cubic or different than the requested value of *hgrid* by a specified *tolerance*. Note that using non-cubic SRD bins can lead to undetermined behavior when rotating the velocities of SRD particles, hence LAMMPS tries to protect you from this problem.

LAMMPS attempts to set the SRD bin size to exactly *hgrid*. However, there must be an integer number of bins in each dimension of the simulation box. Thus the actual bin size will depend on the size and shape of the overall simulation box. The actual bin size is printed as part of the SRD output when a simulation begins.

If the actual bin size is non-cubic by an amount exceeding the tolerance, an error or warning is printed, depending on the style of the *cubic* keyword. Likewise, if the actual bin size differs from the requested *hgrid* value by an amount exceeding the tolerance, then an error or warning is printed. The *tolerance* is a fractional difference. E.g. a tolerance setting of 0.01 on the shape means that if the ratio of any 2 bin dimensions exceeds (1 +/- tolerance) then an error or warning is generated. Similarly, if the ratio of any bin dimension with *hgrid* exceeds (1 +/- tolerance), then an error or warning is generated.

Note

The *fix srd* command can be used with simulations where the size and/or shape of the simulation box changes. This can be due to non-periodic boundary conditions or the use of fixes such as the *fix deform* or *fix wall/srd* commands to impose a shear on an SRD fluid or an interaction with an external wall. If the box size changes then the size of SRD bins must be recalculated every reneighboring. This is not necessary if only the box shape changes. This re-binning is always done so as to fit an integer number of bins in the current box dimension, whether it be a fixed, shrink-wrapped, or periodic boundary, as set by the *boundary* command. If the box size or shape changes, then the size of the search bins must be recalculated every reneighboring. Note that changing the SRD bin size may alter the properties of the SRD fluid, such as its viscosity.

The *shift* keyword determines whether the coordinates of SRD particles are randomly shifted when binned for purposes of rotating their velocities. When no shifting is performed, SRD particles are binned and the velocity distribution of the set of SRD particles in each bin is adjusted via a rotation operator. This is a statistically valid operation if SRD particles move sufficiently far between successive rotations. This is determined by their mean-free path λ . If λ is less than 0.6 of the SRD bin size, then shifting is required. A shift means that all of the SRD particles are shifted by a vector whose coordinates are chosen randomly in the range [-1/2 bin size, 1/2 bin size]. Note that all particles are shifted by the same vector. The specified random number *shiftseed* is used to generate these vectors. This operation sufficiently randomizes which SRD particles are in the same bin, even if *lambda* is small.

If the *shift* flag is set to *no*, then no shifting is performed, but bin data will be communicated if bins overlap processor boundaries. An error will be generated if $\lambda < 0.6$ of the SRD bin size. If the *shift* flag is set to *possible*, then shifting is performed only if $\lambda < 0.6$ of the SRD bin size. A warning is generated to let you know this is occurring. If the *shift* flag is set to *yes* then shifting is performed regardless of the magnitude of λ . Note that the *shiftseed* is not used if the *shift* flag is set to *no*, but must still be specified.

Note that shifting of SRD coordinates requires extra communication, hence it should not normally be enabled unless required.

The *tstat* keyword will thermostat the SRD particles to the specified *Tsrd*. This is done every N timesteps, during the velocity rotation operation, by rescaling the thermal velocity of particles in each SRD bin to the desired temperature. If there is a streaming velocity associated with the system, e.g. due to use of the *fix deform* command to perform a simulation undergoing shear, then that is also accounted for. The mean velocity of each bin of SRD particles is set to the position-dependent streaming velocity, based on the coordinates of the center of the SRD bin. Note that collisions of SRD particles with big particles or walls has a thermostating effect on the colliding particles, so it may not be necessary to thermostat the SRD particles on a bin by bin basis in that case. Also note that for streaming simulations, if no thermostating is performed (the default), then it may take a long time for the SRD fluid to come to equilibrium with a velocity profile that matches the simulation box deformation.

The *rescale* keyword enables rescaling of an SRD particle's velocity if it would travel more than 4 mean-free paths in an

SRD timestep. If an SRD particle exceeds this velocity it is possible it will be lost when migrating to other processors or that collisions with big particles will be missed, either of which will generate errors. Thus the safest mode is to run with rescaling enabled. However rescaling removes kinetic energy from the system (the particle's velocity is reduced). The latter will not typically be a problem if thermostating is enabled via the *tstat* keyword or if SRD collisions with big particles or walls effectively thermostat the system. If you wish to turn off rescaling (on is the default), e.g. for a pure SRD system with no thermostating so that the temperature does not decline over time, the *rescale* keyword can be used. The *no* value turns rescaling off during collisions and the per-bin velocity rotation operation. The *collide* and *rotate* values turn it on for one of the operations and off for the other.

Note

This fix is normally used for simulations with a huge number of SRD particles relative to the number of big particles, e.g. 100 to 1. In this scenario, computations that involve only big particles (neighbor list creation, communication, time integration) can slow down dramatically due to the large number of background SRD particles.

Three other input script commands will largely overcome this effect, speeding up an SRD simulation by a significant amount. These are the *atom_modify first*, *neigh_modify include*, and *comm_modify group* commands. Each takes a group-ID as an argument, which in this case should be the group-ID of the big solute particles.

Additionally, when a *pair_style* for big/big particle interactions is specified, the *pair_coeff* command should be used to turn off big/SRD interactions, e.g. by setting their epsilon or cutoff length to 0.0.

The “*delete_atoms overlap*” command may be useful in setting up an SRD simulation to ensure there are no initial overlaps between big and SRD particles.

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

No information about this fix is written to *binary restart files*. None of the *fix_modify* options are relevant to this fix.

This fix tabulates several SRD statistics which are stored in a vector of length 12, which can be accessed by various *output commands*. The vector values calculated by this fix are “intensive”, meaning they do not scale with the size of the simulation. Technically, the first 8 do scale with the size of the simulation, but treating them as intensive means they are not scaled when printed as part of thermodynamic output.

These are the 12 quantities. All are values for the current timestep, except for quantity 5 and the last three, each of which are cumulative quantities since the beginning of the run.

- (1) # of SRD/big collision checks performed
- (2) # of SRDs which had a collision
- (3) # of SRD/big collisions (including multiple bounces)
- (4) # of SRD particles inside a big particle
- (5) # of SRD particles whose velocity was rescaled to be < Vmax
- (6) # of bins for collision searching
- (7) # of bins for SRD velocity rotation
- (8) # of bins in which SRD temperature was computed
- (9) SRD temperature
- (10) # of SRD particles which have undergone max # of bounces

- (11) max # of bounces any SRD particle has had in a single step
- (12) # of reneighborings due to SRD particles moving too far

No parameter of this fix can be used with the *start/stop* keywords of the [run](#) command. This fix is not invoked during *energy minimization*.

2.224.5 Restrictions

This command can only be used if LAMMPS was built with the SRD package. See the [Build package](#) doc page for more info.

2.224.6 Related commands

[fix wall/srd](#)

2.224.7 Default

The option defaults are: *lambda* (λ) is inferred from *Tsrd*, collision = noslip, overlap = no, inside = error, exact = yes, radius = 1.0, bounce = 0, search = hgrid, cubic = error 0.01, shift = no, tstat = no, and rescale = yes.

(Hecht) Hecht, Harting, Ihle, Herrmann, Phys Rev E, 72, 011408 (2005).

(Petersen) Petersen, Lechman, Plimpton, Grest, in' t Veld, Schunk, J Chem Phys, 132, 174106 (2010).

(Bolintineanu1) Bolintineanu, Lechman, Plimpton, Grest, Phys Rev E, 86, 066703 (2012).

(Bolintineanu2) Bolintineanu, Grest, Lechman, Pierce, Plimpton, Schunk, Comp Particle Mechanics, 1, 321-356 (2014).

2.225 fix store/force command

2.225.1 Syntax

```
fix ID group-ID store/force
```

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

2.225.2 Examples

```
fix 1 all store/force
```

2.225.3 Description

Store the forces on atoms in the group at the point during each timestep when the fix is invoked, as described below. This is useful for storing forces before constraints or other boundary conditions are computed which modify the forces, so that unmodified forces can be [written to a dump file](#) or accessed by other [output commands](#) that use per-atom quantities.

This fix is invoked at the point in the velocity-Verlet timestepping immediately after [pair](#), [bond](#), [angle](#), [dihedral](#), [improper](#), and [long-range](#) forces have been calculated. It is the point in the timestep when various fixes that compute constraint forces are calculated and potentially modify the force on each atom. Examples of such fixes are [fix shake](#), [fix wall](#), and [fix indent](#).

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.

2.225.4 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](#).

2.225.5 Restrictions

none

2.225.6 Related commands

[fix store_state](#)

2.225.7 Default

none

2.226 fix store/state command

2.226.1 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, 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[I], f_ID, f_ID[I], v_name,
d_name, i_name, i2_name[I], d2_name[I],

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
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 dipolar atom
mu = magnitued 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[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
i_name = custom integer vector with name
d_name = custom floating point vector with name
i2_name[I] = Ith column of custom integer array with name
d2_name[I] = Ith column of custom floating-point array with name

- zero or more keyword/value pairs may be appended

- keyword = *com*

com value = yes or no

2.226.2 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
```

2.226.3 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.

If N is not zero, then the attributes will be updated every N steps.

Note

Actually, only atom attributes specified by keywords like xu or vy or $radius$ 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.

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.

2.226.4 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.

Warning

When reading data from a restart file, this fix command has to be specified **exactly** the same way as before. LAMMPS will only check whether a fix is of the same style and has the same fix ID and in case of a match will then try to initialize the fix with the data stored in the binary restart file. If the fix store/state command does not match exactly, data can be corrupted or LAMMPS may crash.

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](#).

2.226.5 Restrictions

none

2.226.6 Related commands

dump custom, compute property/atom, fix property/atom, variable

2.226.7 Default

The option default is com = no.

2.227 fix temp/berendsen command

Accelerator Variants: *temp/berendsen/kk*

2.227.1 Syntax

```
fix ID group-ID temp/berendsen Tstart Tstop Tdamp
```

- ID, group-ID are documented in [fix](#) command
 - temp/berendsen = style name of this fix command
 - Tstart,Tstop = desired temperature at start/end of run
- Tstart can be a variable (see below)
- Tdamp = temperature damping parameter (time units)

2.227.2 Examples

```
fix 1 all temp/berendsen 300.0 300.0 100.0
```

2.227.3 Description

Reset the temperature of a group of atoms by using a Berendsen thermostat ([Berendsen](#)), which rescales their velocities every timestep.

The thermostat is applied to only the translational degrees of freedom for the particles, which is an important consideration for finite-size particles which have rotational degrees of freedom being thermostatted with this fix. The translational degrees of freedom can also have a bias velocity removed from them before thermostating takes place; see the description below.

The desired temperature at each timestep is a ramped value during the run from *Tstart* to *Tstop*. The *Tdamp* parameter is specified in time units and determines how rapidly the temperature is relaxed. For example, a value of 100.0 means to relax the temperature in a timespan of (roughly) 100 time units (tau or fs or ps - see the [units](#) command).

Tstart can be specified as an equal-style [variable](#). In this case, the *Tstop* setting is ignored. 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 target temperature.

Note

This thermostat will generate an error if the current temperature is zero at the end of a timestep. It cannot rescale a zero temperature.

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 temperature.

Note

Unlike the [fix nvt](#) command which performs Nose/Hoover thermostatting AND time integration, this fix does NOT perform time integration. It only modifies velocities to effect thermostatting. Thus you must use a separate time integration fix, like [fix nve](#) to actually update the positions of atoms using the modified velocities. Likewise, this fix should not normally be used on atoms that also have their temperature controlled by another fix - e.g. by [fix nvt](#) or [fix langevin](#) commands.

See the [Howto thermostat](#) page for a discussion of different ways to compute temperature and perform thermostatting.

This fix computes a temperature each timestep. To do this, the fix creates its own compute of style “temp”, as if this command had been issued:

```
compute fix-ID temp group-ID temp
```

See the [compute temp](#) command for details. Note that the ID of the new compute is the fix-ID + underscore + “temp”, and the group for the new compute is the same as the fix group.

Note that this is NOT the compute used by thermodynamic output (see the [thermo_style](#) command) with ID = *thermo_temp*. This means you can change the attributes of this fix’s temperature (e.g. its degrees-of-freedom) via the [compute_modify](#) command or print this temperature during thermodynamic output via the [thermo_style custom](#) command using the appropriate compute-ID. It also means that changing attributes of *thermo_temp* will have no effect on this fix.

Like other fixes that perform thermostatting, this fix can be used with [compute commands](#) that remove a “bias” from the atom velocities. E.g. to apply the thermostat only to atoms within a spatial [region](#), or to remove the center-of-mass velocity from a group of atoms, or to remove the x-component of velocity from the calculation.

This is not done by default, but only if the [fix_modify](#) command is used to assign a temperature compute to this fix that includes such a bias term. See the doc pages for individual [compute temp commands](#) to determine which ones include a bias. In this case, the thermostat works in the following manner: bias is removed from each atom, thermostatting is performed on the remaining thermal degrees of freedom, and the bias is added back in.

Styles with a *gpu*, *intel*, *kk*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed on the [Accelerator packages](#) page. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the GPU, INTEL, KOKKOS, OPENMP, and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the [Build package](#) page for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the [-suffix command-line switch](#) when you invoke LAMMPS, or you can use the [suffix](#) command in your input script.

See the [Accelerator packages](#) page for more instructions on how to use the accelerated styles effectively.

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

This fix writes the cumulative global energy change to [binary restart files](#). See the [read_restart](#) command for info on how to re-specify a fix in an input script that reads a restart file, so that the fix continues in an uninterrupted fashion.

The [fix_modify temp](#) option is supported by this fix. You can use it to assign a temperature [compute](#) you have defined to this fix which will be used in its thermostating procedure, as described above. For consistency, the group used by this fix and by the compute should be the same.

The cumulative energy change in the system imposed by this fix is included in the [thermodynamic output](#) keywords [ecouple](#) and [econserve](#). See the [thermo_style](#) doc page for details.

This fix computes a global scalar which can be accessed by various [output commands](#). The scalar is the same cumulative energy change due to this fix described in the previous paragraph. The scalar value calculated by this fix is “extensive”.

This fix can ramp its target temperature over multiple runs, using the [start](#) and [stop](#) keywords of the [run](#) command. See the [run](#) command for details of how to do this.

This fix is not invoked during [energy minimization](#).

2.227.5 Restrictions

This fix can be used with dynamic groups as defined by the [group](#) command. Likewise it can be used with groups to which atoms are added or deleted over time, e.g. a deposition simulation. However, the conservation properties of the thermostat and barostat are defined for systems with a static set of atoms. You may observe odd behavior if the atoms in a group vary dramatically over time or the atom count becomes very small.

2.227.6 Related commands

[fix nve](#), [fix nvt](#), [fix temp/rescale](#), [fix langevin](#), [fix_modify](#), [compute temp](#), [fix press/berendsen](#)

2.227.7 Default

none

(Berendsen) Berendsen, Postma, van Gunsteren, DiNola, Haak, J Chem Phys, 81, 3684 (1984).

2.228 fix temp/csvr command

2.229 fix temp/csld command

2.229.1 Syntax

```
fix ID group-ID temp/csvr Tstart Tstop Tdamp seed
fix ID group-ID temp/csld Tstart Tstop Tdamp seed
```

- ID, group-ID are documented in [fix](#) command
 - temp/csvr or temp/csld = style name of this fix command
 - Tstart,Tstop = desired temperature at start/end of run
- Tstart can be a variable (see below)
- Tdamp = temperature damping parameter (time units)
 - seed = random number seed to use for white noise (positive integer)

2.229.2 Examples

```
fix 1 all temp/csvr 300.0 300.0 100.0 54324
fix 1 all temp/csld 100.0 300.0 10.0 123321
```

2.229.3 Description

Adjust the temperature with a canonical sampling thermostat that uses global velocity rescaling with Hamiltonian dynamics (*temp/csvr*) ([Bussi1](#)), or Langevin dynamics (*temp/csld*) ([Bussi2](#)). In the case of *temp/csvr* the thermostat is similar to the empirical Berendsen thermostat in [temp/berendsen](#), but chooses the actual scaling factor from a suitably chosen (gaussian) distribution rather than having it determined from the time constant directly. In the case of *temp/csld* the velocities are updated to a linear combination of the current velocities with a gaussian distribution of velocities at the desired temperature. Both thermostats are applied every timestep.

The thermostat is applied to only the translational degrees of freedom for the particles, which is an important consideration for finite-size particles which have rotational degrees of freedom being thermostatted with these fixes. The translational degrees of freedom can also have a bias velocity removed from them before thermostating takes place; see the description below.

The desired temperature at each timestep is a ramped value during the run from *Tstart* to *Tstop*. The *Tdamp* parameter is specified in time units and determines how rapidly the temperature is relaxed. For example, a value of 100.0 means to relax the temperature in a timespan of (roughly) 100 time units (tau or fs or ps - see the [units](#) command).

Tstart can be specified as an equal-style [variable](#). In this case, the *Tstop* setting is ignored. 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 target temperature.

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 temperature.

Note

Unlike the [fix nvt](#) command which performs Nose/Hoover thermostatting AND time integration, these fixes do NOT perform time integration. They only modify velocities to effect thermostatting. Thus you must use a separate time integration fix, like [fix nve](#) to actually update the positions of atoms using the modified velocities. Likewise, these fixes should not normally be used on atoms that also have their temperature controlled by another fix - e.g. by [fix nvt](#) or [fix langevin](#) commands.

See the [Howto thermostat](#) page for a discussion of different ways to compute temperature and perform thermostatting. These fixes compute a temperature each timestep. To do this, the fix creates its own compute of style “temp”, as if this command had been issued:

```
compute fix-ID_temp group-ID temp
```

See the [compute temp](#) command for details. Note that the ID of the new compute is the fix-ID + underscore + “temp”, and the group for the new compute is the same as the fix group.

Note that this is NOT the compute used by thermodynamic output (see the [thermo_style](#) command) with ID = *thermo_temp*. This means you can change the attributes of this fix’s temperature (e.g. its degrees-of-freedom) via the [compute_modify](#) command or print this temperature during thermodynamic output via the [thermo_style custom](#) command using the appropriate compute-ID. It also means that changing attributes of *thermo_temp* will have no effect on this fix.

Like other fixes that perform thermostatting, this fix can be used with [compute commands](#) that remove a “bias” from the atom velocities. E.g. to apply the thermostat only to atoms within a spatial [region](#), or to remove the center-of-mass velocity from a group of atoms, or to remove the x-component of velocity from the calculation.

This is not done by default, but only if the [fix_modify](#) command is used to assign a temperature compute to this fix that includes such a bias term. See the doc pages for individual [compute temp commands](#) to determine which ones include a bias. In this case, the thermostat works in the following manner: bias is removed from each atom, thermostatting is performed on the remaining thermal degrees of freedom, and the bias is added back in.

An important feature of these thermostats is that they have an associated effective energy that is a constant of motion. The effective energy is the total energy (kinetic + potential) plus the accumulated kinetic energy changes due to the thermostat. The latter quantity is the global scalar computed by these fixes. This feature is useful to check the integration of the equations of motion against discretization errors. In other words, the conservation of the effective energy can be used to choose an appropriate integration [timestep](#). This is similar to the usual paradigm of checking the conservation of the total energy in the microcanonical ensemble.

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

These fixes write the cumulative global energy change and the random number generator states to [binary restart files](#). See the [read_restart](#) command for info on how to re-specify a fix in an input script that reads a restart file, so that the selected fix continues in an uninterrupted fashion. The random number generator state can only be restored when the number of processors remains unchanged from what is recorded in the restart file.

The [fix_modify temp](#) option is supported by these fixes. You can use it to assign a temperature [compute](#) you have defined to these fixes which will be used in its thermostatting procedure, as described above. For consistency, the group used by these fixes and by the compute should be the same.

The cumulative energy change in the system imposed by these fixes is included in the [thermodynamic output](#) keywords *ecouple* and *econserve*. See the [thermo_style](#) doc page for details.

These fixes compute a global scalar which can be accessed by various *output commands*. The scalar is the same cumulative energy change due to this fix described in the previous paragraph. The scalar value calculated by this fix is “extensive”.

These fixes can ramp their target temperature over multiple runs, using the *start* and *stop* keywords of the *run* command. See the *run* command for details of how to do this.

These fixes are not invoked during *energy minimization*.

2.229.5 Restrictions

Fix *temp/csld* is not compatible with *fix shake*.

These fixes are part of the EXTRA-FIX package. They are only enabled if LAMMPS was built with that package. See the *Build package* page for more info.

These fixes can be used with dynamic groups as defined by the *group* command. Likewise it can be used with groups to which atoms are added or deleted over time, e.g. a deposition simulation. However, the conservation properties of the thermostat and barostat are defined for systems with a static set of atoms. You may observe odd behavior if the atoms in a group vary dramatically over time or the atom count becomes very small.

2.229.6 Related commands

fix nve, fix nvt, fix temp/rescale, fix langevin, fix_modify, compute temp, fix temp/berendsen

2.229.7 Default

none

(Bussi1) Bussi, Donadio and Parrinello, J. Chem. Phys. 126, 014101(2007)

(Bussi2) Bussi and Parrinello, Phys. Rev. E 75, 056707 (2007)

2.230 fix temp/rescale command

Accelerator Variants: *temp/rescale/kk*

2.230.1 Syntax

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

- ID, group-ID are documented in *fix* command
- temp/rescale = style name of this fix command
- N = perform rescaling every N steps
- Tstart,Tstop = desired temperature at start/end of run (temperature units)

Tstart can be a variable (see below)

- window = only rescale if temperature is outside this window (temperature units)

- fraction = rescale to target temperature by this fraction

2.230.2 Examples

```
fix 3 flow temp/rescale 100 1.0 1.1 0.02 0.5
fix 3 boundary temp/rescale 1 1.0 1.5 0.05 1.0
fix 3 boundary temp/rescale 1 1.0 1.5 0.05 1.0
```

2.230.3 Description

Reset the temperature of a group of atoms by explicitly rescaling their velocities.

The rescaling is applied to only the translational degrees of freedom for the particles, which is an important consideration if finite-size particles which have rotational degrees of freedom are being thermostatted with this fix. The translational degrees of freedom can also have a bias velocity removed from them before thermostatting takes place; see the description below.

Rescaling is performed every N timesteps. The target temperature is a ramped value between the *Tstart* and *Tstop* temperatures at the beginning and end of the run.

Note

This thermostat will generate an error if the current temperature is zero at the end of a timestep it is invoked on. It cannot rescale a zero temperature.

Tstart can be specified as an equal-style *variable*. In this case, the *Tstop* setting is ignored. 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 target temperature.

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 temperature.

Rescaling is only performed if the difference between the current and desired temperatures is greater than the *window* value. The amount of rescaling that is applied is a *fraction* (from 0.0 to 1.0) of the difference between the actual and desired temperature. E.g. if *fraction* = 1.0, the temperature is reset to exactly the desired value.

Note

Unlike the *fix nvt* command which performs Nose/Hoover thermostatting AND time integration, this fix does NOT perform time integration. It only modifies velocities to effect thermostatting. Thus you must use a separate time integration fix, like *fix nve* to actually update the positions of atoms using the modified velocities. Likewise, this fix should not normally be used on atoms that also have their temperature controlled by another fix - e.g. by *fix nvt* or *fix langevin* commands.

See the *Howto thermostat* page for a discussion of different ways to compute temperature and perform thermostatting.

This fix computes a temperature each timestep. To do this, the fix creates its own compute of style “temp”, as if one of this command had been issued:

```
compute fix-ID_temp group-ID temp
```

See the [compute temp](#) for details. Note that the ID of the new compute is the fix-ID + underscore + “temp”, and the group for the new compute is the same as the fix group.

Note that this is NOT the compute used by thermodynamic output (see the [thermo_style](#) command) with ID = *thermo_temp*. This means you can change the attributes of this fix’s temperature (e.g. its degrees-of-freedom) via the [compute_modify](#) command or print this temperature during thermodynamic output via the [thermo_style custom](#) command using the appropriate compute-ID. It also means that changing attributes of *thermo_temp* will have no effect on this fix.

Like other fixes that perform thermostating, this fix can be used with [compute commands](#) that remove a “bias” from the atom velocities. E.g. to apply the thermostat only to atoms within a spatial [region](#), or to remove the center-of-mass velocity from a group of atoms, or to remove the x-component of velocity from the calculation.

This is not done by default, but only if the [fix_modify](#) command is used to assign a temperature compute to this fix that includes such a bias term. See the doc pages for individual [compute temp commands](#) to determine which ones include a bias. In this case, the thermostat works in the following manner: bias is removed from each atom, thermostating is performed on the remaining thermal degrees of freedom, and the bias is added back in.

Styles with a *gpu*, *intel*, *kk*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed on the [Accelerator packages](#) page. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the GPU, INTEL, KOKKOS, OPENMP, and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the [Build package](#) page for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the [-suffix command-line switch](#) when you invoke LAMMPS, or you can use the [suffix](#) command in your input script.

See the [Accelerator packages](#) page for more instructions on how to use the accelerated styles effectively.

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

This fix writes the cumulative global energy change to [binary restart files](#). See the [read_restart](#) command for info on how to re-specify a fix in an input script that reads a restart file, so that the fix continues in an uninterrupted fashion.

The [fix_modify temp](#) option is supported by this fix. You can use it to assign a temperature [compute](#) you have defined to this fix which will be used in its thermostating procedure, as described above. For consistency, the group used by this fix and by the compute should be the same.

The cumulative energy change in the system imposed by this fix is included in the [thermodynamic output](#) keywords *ecouple* and *econserve*. See the [thermo_style](#) doc page for details.

This fix computes a global scalar which can be accessed by various [output commands](#). The scalar is the same cumulative energy change due to this fix described in the previous paragraph. The scalar value calculated by this fix is “extensive”.

This fix can ramp its target temperature over multiple runs, using the *start* and *stop* keywords of the [run](#) command. See the [run](#) command for details of how to do this.

This fix is not invoked during [energy minimization](#).

2.230.5 Restrictions

none

2.230.6 Related commands

fix langevin, fix nvt, fix_modify

2.230.7 Default

none

2.231 fix temp/rescale/eff command

2.231.1 Syntax

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

- ID, group-ID are documented in [fix](#) command
- temp/rescale/eff = style name of this fix command
- N = perform rescaling every N steps
- Tstart,Tstop = desired temperature at start/end of run (temperature units)
- window = only rescale if temperature is outside this window (temperature units)
- fraction = rescale to target temperature by this fraction

2.231.2 Examples

```
fix 3 flow temp/rescale/eff 10 1.0 100.0 0.02 1.0
```

2.231.3 Description

Reset the temperature of a group of nuclei and electrons in the [electron force field](#) model by explicitly rescaling their velocities.

The operation of this fix is exactly like that described by the [fix temp/rescale](#) command, except that the rescaling is also applied to the radial electron velocity for electron particles.

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

No information about this fix is written to *binary restart files*.

The *fix_modify temp* option is supported by this fix. You can use it to assign a temperature *compute* you have defined to this fix which will be used in its thermostating procedure, as described above. For consistency, the group used by this fix and by the compute should be the same.

The cumulative energy change in the system imposed by this fix is included in the *thermodynamic output* keywords *ecouple* and *econserve*. See the *thermo_style* doc page for details.

This fix computes a global scalar which can be accessed by various *output commands*. The scalar is the same cumulative energy change due to this fix described in the previous paragraph. The scalar value calculated by this fix is “extensive”.

This fix can ramp its target temperature over multiple runs, using the *start* and *stop* keywords of the *run* command. See the *run* command for details of how to do this.

This fix is not invoked during *energy minimization*.

2.231.5 Restrictions

This fix is part of the EFF package. It is only enabled if LAMMPS was built with that package. See the *Build package* page for more info.

2.231.6 Related commands

fix langevin/eff, *fix nvt/eff*, *fix_modify*, *fix temp rescale*,

2.231.7 Default

none

2.232 fix tfmc command

2.232.1 Syntax

`fix ID group-ID tfmc Delta Temp seed keyword value`

- ID, group-ID are documented in *fix* command
- tfmc = style name of this fix command
- Delta = maximal displacement length (distance units)
- Temp = imposed temperature of the system
- seed = random number seed (positive integer)
- zero or more keyword/arg pairs may be appended
- keyword = *com* or *rot*

com args = xflag yflag zflag
 xflag,yflag,zflag = 0/1 to exclude/include each dimension
 rot args = none

2.232.2 Examples

```
fix 1 all tfmc 0.1 1000.0 159345
fix 1 all tfmc 0.05 600.0 658943 com 1 1 0
fix 1 all tfmc 0.1 750.0 387068 com 1 1 1 rot
```

2.232.3 Description

Perform uniform-acceptance force-bias Monte Carlo (fbMC) simulations, using the time-stamped force-bias Monte Carlo (tfMC) algorithm described in ([Mees](#)) and ([Bal](#)).

One successful use case of force-bias Monte Carlo methods is that they can be used to extend the time scale of atomistic simulations, in particular when long time scale relaxation effects must be considered; some interesting examples are given in the review by ([Neyts](#)). An example of a typical use case would be the modelling of chemical vapor deposition (CVD) processes on a surface, in which impacts by gas-phase species can be performed using MD, but subsequent relaxation of the surface is too slow to be done using MD only. Using tfMC can allow for a much faster relaxation of the surface, so that higher fluxes can be used, effectively extending the time scale of the simulation. (Such an alternating simulation approach could be set up using a [loop](#).)

The initial version of tfMC algorithm in ([Mees](#)) contained an estimation of the effective time scale of such a simulation, but it was later shown that the speed-up one can gain from a tfMC simulation is system- and process-dependent, ranging from none to several orders of magnitude. In general, solid-state processes such as (re)crystallization or growth can be accelerated by up to two or three orders of magnitude, whereas diffusion in the liquid phase is not accelerated at all. The observed pseudodynamics when using the tfMC method is not the actual dynamics one would obtain using MD, but the relative importance of processes can match the actual relative dynamics of the system quite well, provided *Delta* is chosen with care. Thus, the system's equilibrium is reached faster than in MD, along a path that is generally roughly similar to a typical MD simulation (but not necessarily so). See ([Bal](#)) for details.

Each step, all atoms in the selected group are displaced using the stochastic tfMC algorithm, which is designed to sample the canonical (NVT) ensemble at the temperature *Temp*. Although tfMC is a Monte Carlo algorithm and thus strictly speaking does not perform time integration, it is similar in the sense that it uses the forces on all atoms in order to update their positions. Therefore, it is implemented as a time integration fix, and no other fixes of this type (such as [fix nve](#)) should be used at the same time. Because velocities do not play a role in this kind of Monte Carlo simulations, instantaneous temperatures as calculated by [temperature computes](#) or [thermodynamic output](#) have no meaning: the only relevant temperature is the sampling temperature *Temp*. Similarly, performing tfMC simulations does not require setting a [timestep](#) and the [simulated time](#) as calculated by LAMMPS is meaningless.

The critical parameter determining the success of a tfMC simulation is *Delta*, the maximal displacement length of the lightest element in the system: the larger it is, the longer the effective time scale of the simulation will be (there is an approximately quadratic dependence). However, *Delta* must also be chosen sufficiently small in order to comply with detailed balance; in general values between 5 and 10 % of the nearest neighbor distance are found to be a good choice. For a more extensive discussion with specific examples, please refer to ([Bal](#)), which also describes how the code calculates element-specific maximal displacements from *Delta*, based on the fourth root of their mass.

Because of the uncorrelated movements of the atoms, the center-of-mass of the fix group will not necessarily be stationary, just like its orientation. When the *com* keyword is used, all atom positions will be shifted (after every tfMC iteration) in order to fix the position of the center-of-mass along the included directions, by setting the corresponding flag to 1. The *rot* keyword does the same for the rotational component of the tfMC displacements after every iteration.

Note

the *com* and *rot* keywords should not be used if an external force is acting on the specified fix group, along the included directions. This can be either a true external force (e.g. through [fix wall](#)) or forces due to the interaction

with atoms not included in the fix group. This is because in such cases, translations or rotations of the fix group could be induced by these external forces, and removing them will lead to a violation of detailed balance.

2.232.4 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 is not invoked during *energy minimization*.

2.232.5 Restrictions

This fix is part of the MC package. It is only enabled if LAMMPS was built with that package. See the *Build package* doc page for more info.

This fix is not compatible with *fix shake*.

2.232.6 Related commands

fix gcmc, *fix nvt*

2.232.7 Default

The option default is com = 0 0 0

(Bal) K. M Bal and E. C. Neyts, J. Chem. Phys. 141, 204104 (2014).

(Mees) M. J. Mees, G. Pourtois, E. C. Neyts, B. J. Thijssse, and A. Stesmans, Phys. Rev. B 85, 134301 (2012).

(Neyts) E. C. Neyts and A. Bogaerts, Theor. Chem. Acc. 132, 1320 (2013).

2.233 fix tgnvt/drude command

2.234 fix tgnpt/drude command

2.234.1 Syntax

`fix ID group-ID style_name keyword values ...`

- ID, group-ID are documented in *fix* command
- style_name = *tgnvt/drude* or *tgnpt/drude*
- one or more keyword/values pairs may be appended

keyword = temp iso or aniso or tri or x or y or z or xy or yz or xz or couple or tchain or pchain or ↴mtk or tloop or ploop or nreset or scalexy or scaleyz or scalexz or flip or fixedpoint
 temp values = Tstart Tstop Tdamp Tdrude Tdamp_drude
 Tstart, Tstop = external temperature at start/end of run (temperature units)
 Tdamp = temperature damping parameter (time units)
 Tdrude = desired temperature of Drude oscillators (temperature units)
 Tdamp_drude = temperature damping parameter for Drude oscillators (time units)
 iso or aniso or tri values = Pstart Pstop Pdamp
 Pstart,Pstop = scalar external pressure at start/end of run (pressure units)
 Pdamp = pressure damping parameter (time units)
 x or y or z or xy or yz or xz values = Pstart Pstop Pdamp
 Pstart,Pstop = external stress tensor component at start/end of run (pressure units)
 Pdamp = stress damping parameter (time units)
 couple = none or xyz or xy or yz or xz
 tchain value = N
 N = length of thermostat chain (1 = single thermostat)
 pchain value = N
 N length of thermostat chain on barostat (0 = no thermostat)
 mtk value = yes or no = add in MTK adjustment term or not
 tloop value = M
 M = number of sub-cycles to perform on thermostat
 ploop value = M
 M = number of sub-cycles to perform on barostat thermostat
 nreset value = reset reference cell every this many timesteps
 scalexy value = yes or no = scale xy with ly
 scaleyz value = yes or no = scale yz with lz
 scalexz value = yes or no = scale xz with lz
 flip value = yes or no = allow or disallow box flips when it becomes highly skewed
 fixedpoint values = x y z
 x,y,z = perform barostat dilation/contraction around this point (distance units)

2.234.2 Examples

```

comm_modify vel yes
fix 1 all tgnvt/drude temp 300.0 300.0 100.0 1.0 20.0
fix 1 water tgnpt/drude temp 300.0 300.0 100.0 1.0 20.0 iso 0.0 0.0 1000.0
fix 2 jello tgnpt/drude temp 300.0 300.0 100.0 1.0 20.0 tri 5.0 5.0 1000.0
fix 2 ice tgnpt/drude temp 250.0 250.0 100.0 1.0 20.0 x 1.0 1.0 0.5 y 2.0 2.0 0.5 z 3.0 3.0 0.5 yz 0.1 0.1 0.5
  ↴xz 0.2 0.2 0.5 xy 0.3 0.3 0.5 nreset 1000
  
```

Example input scripts available: examples/PACKAGES/drude

2.234.3 Description

These commands are variants of the Nose-Hoover fix styles [fix nvt](#) and [fix npt](#) for thermalized Drude polarizable models. They apply temperature-grouped Nose-Hoover thermostat (TGNH) proposed by ([Son](#)). When there are fast vibrational modes with frequencies close to Drude oscillators (e.g. double bonds or out-of-plane torsions), this thermostat can provide better kinetic energy equipartitioning.

The difference between TGNH and the original Nose-Hoover thermostat is that, TGNH separates the kinetic energy of the group into three contributions: molecular center of mass (COM) motion, motion of COM of atom-Drude pairs or non-polarizable atoms relative to molecular COM, and relative motion of atom-Drude pairs. An independent Nose-Hoover chain is applied to each type of motion. The temperatures for these three types of motion are denoted as

molecular translational temperature (T_M), real atomic temperature (T_R) and Drude temperature (T_D), which are defined in terms of their associated degrees of freedom (DOF):

$$T_M = \frac{\sum_i^{N_{\text{mol}}} M_i V_i^2}{3 \left(N_{\text{mol}} - \frac{N_{\text{mol}}}{N_{\text{mol,sys}}} \right) k_B}$$

$$T_R = \frac{\sum_i^{N_{\text{real}}} m_i (v_i - v_{M,i})^2}{(N_{\text{DOF}} - 3N_{\text{mol}} + 3 \frac{N_{\text{mol}}}{N_{\text{mol,sys}}} - 3N_{\text{drude}}) k_B}$$

$$T_D = \frac{\sum_i^{N_{\text{drude}}} m'_i v'_i^2}{3N_{\text{drude}} k_B}$$

Here N_{mol} and $N_{\text{mol,sys}}$ are the numbers of molecules in the group and in the whole system, respectively. N_{real} is the number of atom-Drude pairs and non-polarizable atoms in the group. N_{drude} is the number of Drude particles in the group. N_{DOF} is the DOF of the group. M_i and V_i are the mass and the COM velocity of the i-th molecule. m_i is the mass of the i-th atom-Drude pair or non-polarizable atom. v_i is the velocity of COM of i-th atom-Drude pair or non-polarizable atom. $v_{M,i}$ is the COM velocity of the molecule the i-th atom-Drude pair or non-polarizable atom belongs to. m'_i and v'_i are the reduced mass and the relative velocity of the i-th atom-Drude pair.

Note

These fixes require that each atom knows whether it is a Drude particle or not. You must therefore use the [fix drude](#) command to specify the Drude status of each atom type.

Because the TGNH thermostat thermostats the molecular COM motion, all atoms belonging to the same molecule must be in the same group. That is, these fixes can not be applied to a subset of a molecule.

For this fix to act correctly, ghost atoms need to know their velocity. You must use the [comm_modify](#) command to enable this.

These fixes assume that the translational DOF of the whole system is removed. It is therefore recommended to invoke [fix momentum](#) command so that the T_M is calculated correctly.

The thermostat parameters are specified using the `temp` keyword. The thermostat is applied to only the translational DOF for the particles. The translational DOF can also have a bias velocity removed before thermostating takes place; see the description below. The desired temperature for molecular and real atomic motion is a ramped value during the run from `Tstart` to `Tstop`. The `Tdamp` parameter is specified in time units and determines how rapidly the temperature is relaxed. For example, a value of 10.0 means to relax the temperature in a timespan of (roughly) 10 time units (e.g. τ or fs or ps - see the [units](#) command). The parameter `Tdrude` is the desired temperature for Drude motion at each timestep. Similar to `Tdamp`, the `Tdamp_drude` parameter determines the relaxation speed for Drude motion. Fix group are the only ones whose velocities and positions are updated by the velocity/position update portion of the integration. Other thermostat-related keywords are `tchain` and `tloop`, which are detailed in [fix nvt](#).

Note

A Nose-Hoover thermostat will not work well for arbitrary values of `Tdamp`. If `Tdamp` is too small, the temperature can fluctuate wildly; if it is too large, the temperature will take a very long time to equilibrate. A good choice for many models is a `Tdamp` of around 100 timesteps. A smaller `Tdamp_drude` value would be required to maintain Drude motion at low temperature.

```
fix 1 all nvt temp 300.0 300.0 $(100.0*dt) 1.0 $(20.0*dt)
```

The barostat parameters for fix style *tgnpt/drude* is specified using one or more of the *iso*, *aniso*, *tri*, *x*, *y*, *z*, *xy*, *xz*, *yz*, and *couple* keywords. These keywords give you the ability to specify all 6 components of an external stress tensor, and to couple various of these components together so that the dimensions they represent are varied together during a constant-pressure simulation. Other barostat-related keywords are *pchain*, *mtk*, *ploop*, *nreset*, *scalexy*, *scaleyz*, *scalexz*, *flip* and *fixedpoint*. The meaning of barostat parameters are detailed in [fix npt](#).

Regardless of what atoms are in the fix group (the only atoms which are time integrated), a global pressure or stress tensor is computed for all atoms. Similarly, when the size of the simulation box is changed, all atoms are re-scaled to new positions.

Note

Unlike the [fix temp/berendsen](#) command which performs thermostating but NO time integration, these fixes perform thermostating/barostating AND time integration. Thus you should not use any other time integration fix, such as [fix nve](#) on atoms to which this fix is applied. Likewise, these fixes should not be used on atoms that also have their temperature controlled by another fix - e.g. by [fix langevin/drude](#) command.

See the [Howto thermostat](#) and [Howto barostat](#) doc pages for a discussion of different ways to compute temperature and perform thermostating and barostating.

Like other fixes that perform thermostating, this fix can be used with [compute commands](#) that remove a “bias” from the atom velocities. E.g. to apply the thermostat only to atoms within a spatial *region*, or to remove the center-of-mass velocity from a group of atoms, or to remove the x-component of velocity from the calculation.

This is not done by default, but only if the [fix_modify](#) command is used to assign a temperature compute to this fix that includes such a bias term. See the doc pages for individual [compute temp commands](#) to determine which ones include a bias. In this case, the thermostat works in the following manner: bias is removed from each atom, thermostating is performed on the remaining thermal degrees of freedom, and the bias is added back in.

Note

However, not all temperature compute commands are valid to be used with these fixes. Precisely, only temperature compute that does not modify the DOF of the group can be used. E.g. [compute temp/ramp](#) and [compute viscosity/cos](#) compute the kinetic energy after remove a velocity gradient without affecting the DOF of the group, then they can be invoked in this way. In contrast, [compute temp/partial](#) may remove the DOF at one or more dimensions, therefore it cannot be used with these fixes.

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

These fixes writes the state of all the thermostat and barostat variables to [binary restart files](#). See the [read_restart](#) command for info on how to re-specify a fix in an input script that reads a restart file, so that the operation of the fix continues in an uninterrupted fashion.

The [fix_modify](#) *temp* and *press* options are supported by these fixes. You can use them to assign a [compute](#) you have defined to this fix which will be used in its thermostating or barostating procedure, as described above. If you do this, note that the kinetic energy derived from the compute temperature should be consistent with the virial term computed using all atoms for the pressure. LAMMPS will warn you if you choose to compute temperature on a subset of atoms.

Note

If both the *temp* and *press* keywords are used in a single *thermo_modify* command (or in two separate commands), then the order in which the keywords are specified is important. Note that a *pressure compute* defines its own temperature compute as an argument when it is specified. The *temp* keyword will override this (for the pressure compute being used by *fix npt*), but only if the *temp* keyword comes after the *press* keyword. If the *temp* keyword comes before the *press* keyword, then the new pressure compute specified by the *press* keyword will be unaffected by the *temp* setting.

The cumulative energy change in the system imposed by these fixes, due to thermostatting and/or barostatting, are included in the *thermodynamic output* keywords *ecouple* and *econserve*. See the *thermo_style* page for details.

These fixes compute a global scalar which can be accessed by various *output commands*. The scalar is the same cumulative energy change due to this fix described in the previous paragraph. The scalar value calculated by this fix is “extensive”.

These fixes also compute a global vector of quantities, which can be accessed by various *output commands*. The vector values are “intensive”. The vector stores the three temperatures T_M , T_R and T_D .

These fixes can ramp their external temperature and pressure over multiple runs, using the *start* and *stop* keywords of the *run* command. See the *run* command for details of how to do this.

These fixes are not invoked during *energy minimization*.

2.234.5 Restrictions

These fixes are only available when LAMMPS was built with the DRUDE package. These fixes cannot be used with dynamic groups as defined by the *group* command. These fixes cannot be used in 2D simulations.

X , y , z cannot be barostatted if the associated dimension is not periodic. Xy , xz , and yz can only be barostatted if the simulation domain is triclinic and the second dimension in the keyword (y dimension in xy) is periodic. The *create_box*, *read data*, and *read_restart* commands specify whether the simulation box is orthogonal or non-orthogonal (triclinic) and explain the meaning of the xy, xz, yz tilt factors.

For the *temp* keyword, the final T_{stop} cannot be 0.0 since it would make the external $T = 0.0$ at some timestep during the simulation which is not allowed in the Nose/Hoover formulation.

The *scaleyz yes*, *scalexz yes*, and *scalexy yes* options can only be used if the second dimension in the keyword is periodic, and if the tilt factor is not coupled to the barostat via keywords *tri*, yz , xz , and xy .

2.234.6 Related commands

fix drude, *fix nvt*, *fix_npt*, *fix_modify*

2.234.7 Default

The keyword defaults are tchain = 3, pchain = 3, mtk = yes, tloop = 1, ploop = 1, nreset = 0, couple = none, flip = yes, scaleyz = scalexz = scalexy = yes if periodic in second dimension and not coupled to barostat, otherwise no.

(Son) Son, McDaniel, Cui and Yethiraj, J Phys Chem Lett, 10, 7523 (2019).

2.235 fix thermal/conductivity command

2.235.1 Syntax

```
fix ID group-ID thermal/conductivity N edim Nbin keyword value ...
```

- ID, group-ID are documented in [fix](#) command
 - thermal/conductivity = style name of this fix command
 - N = perform kinetic energy exchange every N steps
 - edim = x or y or z = direction of kinetic energy transfer
 - Nbin = # of layers in edim direction (must be even number)
 - zero or more keyword/value pairs may be appended
 - keyword = *swap*
- swap value = Nswap = number of swaps to perform every N steps

2.235.2 Examples

```
fix 1 all thermal/conductivity 100 z 20
fix 1 all thermal/conductivity 50 z 20 swap 2
```

2.235.3 Description

Use the Muller-Plathe algorithm described in [this paper](#) to exchange kinetic energy between two particles in different regions of the simulation box every N steps. This induces a temperature gradient in the system. As described below this enables the thermal conductivity of a material to be calculated. This algorithm is sometimes called a reverse non-equilibrium MD (reverse NEMD) approach to computing thermal conductivity. This is because the usual NEMD approach is to impose a temperature gradient on the system and measure the response as the resulting heat flux. In the Muller-Plathe method, the heat flux is imposed, and the temperature gradient is the system's response.

See the [compute heat/flux](#) command for details on how to compute thermal conductivity in an alternate way, via the Green-Kubo formalism.

The simulation box is divided into *Nbin* layers in the *edim* direction, where the layer 1 is at the low end of that dimension and the layer *Nbin* is at the high end. Every N steps, Nswap pairs of atoms are chosen in the following manner. Only atoms in the fix group are considered. The hottest Nswap atoms in layer 1 are selected. Similarly, the coldest Nswap atoms in the “middle” layer (see below) are selected. The two sets of Nswap atoms are paired up and their velocities are exchanged. This effectively swaps their kinetic energies, assuming their masses are the same. If the masses are different, an exchange of velocities relative to center of mass motion of the two atoms is performed, to conserve kinetic

energy. Over time, this induces a temperature gradient in the system which can be measured using commands such as the following, which writes the temperature profile (assuming $z = edim$) to the file tmp.profile:

```
compute ke all ke/atom
variable temp atom c_ke/1.5
compute layers all chunk/atom bin/1d z lower 0.05 units reduced
fix    3 all ave/chunk 10 100 1000 layers v_temp file tmp.profile
```

Note that by default, $Nswap = 1$, though this can be changed by the optional *swap* keyword. Setting this parameter appropriately, in conjunction with the swap rate N , allows the heat flux to be adjusted across a wide range of values, and the kinetic energy to be exchanged in large chunks or more smoothly.

The “middle” layer for velocity swapping is defined as the $Nbin/2 + 1$ layer. Thus if $Nbin = 20$, the two swapping layers are 1 and 11. This should lead to a symmetric temperature profile since the two layers are separated by the same distance in both directions in a periodic sense. This is why $Nbin$ is restricted to being an even number.

As described below, the total kinetic energy transferred by these swaps is computed by the fix and can be output. Dividing this quantity by time and the cross-sectional area of the simulation box yields a heat flux. The ratio of heat flux to the slope of the temperature profile is proportional to the thermal conductivity of the fluid, in appropriate units. See the [Muller-Plathe paper](#) for details.

Note

If your system is periodic in the direction of the heat flux, then the flux is going in 2 directions. This means the effective heat flux in one direction is reduced by a factor of 2. You will see this in the equations for thermal conductivity (κ) in the Muller-Plathe paper. LAMMPS is simply tallying kinetic energy which does not account for whether or not your system is periodic; you must use the value appropriately to yield a κ for your system.

Note

After equilibration, if the temperature gradient you observe is not linear, then you are likely swapping energy too frequently and are not in a regime of linear response. In this case you cannot accurately infer a thermal conductivity and should try increasing the *Nevery* parameter.

2.235.4 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 cumulative kinetic energy transferred between the bottom and middle of the simulation box (in the *edim* direction) is stored as a scalar quantity by this fix. This quantity is zeroed when the fix is defined and accumulates thereafter, once every N steps. The units of the quantity are energy; see the *units* command for details. 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*.

2.235.5 Restrictions

Swaps conserve both momentum and kinetic energy, even if the masses of the swapped atoms are not equal. Thus you should not need to thermostat the system. If you do use a thermostat, you may want to apply it only to the non-swapped dimensions (other than *vdim*).

LAMMPS does not check, but you should not use this fix to swap the kinetic energy of atoms that are in constrained molecules, e.g. via [fix shake](#) or [fix rigid](#). This is because application of the constraints will alter the amount of transferred momentum. You should, however, be able to use flexible molecules. See the [Zhang paper](#) for a discussion and results of this idea.

When running a simulation with large, massive particles or molecules in a background solvent, you may want to only exchange kinetic energy between solvent particles.

2.235.6 Related commands

[fix ehex](#), [fix heat](#), [fix ave/chunk](#), [fix viscosity](#), [compute heat/flux](#)

2.235.7 Default

The option defaults are swap = 1.

(**Muller-Plathe**) Muller-Plathe, J Chem Phys, 106, 6082 (1997).

(**Zhang**) Zhang, Lussetti, de Souza, Muller-Plathe, J Phys Chem B, 109, 15060-15067 (2005).

2.236 fix ti/spring command

2.236.1 Syntax

```
fix ID group-ID ti/spring k t_s t_eq keyword value ...
```

- ID, group-ID are documented in [fix](#) command
- ti/spring = style name of this fix command
- k = spring constant (force/distance units)
- t_eq = number of steps for the equilibration procedure
- t_s = number of steps for the switching procedure
- zero or more keyword/value pairs may be appended to args
- keyword = *function*

function value = function-ID

function-ID = ID of the switching function (1 or 2)

2.236.2 Example

```
fix 1 all ti/spring 50.0 2000 1000 function 2
```

2.236.3 Description

This fix allows you to compute the free energy of crystalline solids by performing a nonequilibrium thermodynamic integration between the solid of interest and an Einstein crystal. A detailed explanation of how to use this command and choose its parameters for optimal performance and accuracy is given in the paper by [Freitas](#). The paper also presents a short summary of the theory of nonequilibrium thermodynamic integration.

The thermodynamic integration procedure is performed by rescaling the force on each atom. Given an atomic configuration the force (F) on each atom is given by

$$F = (1 - \lambda) F_{\text{solid}} + \lambda F_{\text{harm}}$$

where F_{solid} is the force that acts on an atom due to an interatomic potential (*e.g.* EAM potential), F_{harm} is the force due to the Einstein crystal harmonic spring, and λ is the coupling parameter of the thermodynamic integration. An Einstein crystal is a solid where each atom is attached to its equilibrium position by a harmonic spring with spring constant k . With this fix a spring force is applied independently to each atom in the group defined by the fix to tether it to its initial position. The initial position of each atom is its position at the time the fix command was issued.

The fix acts as follows: during the first t_{eq} steps after the fix is defined the value of λ is zero. This is the period to equilibrate the system in the $\lambda = 0$ state. After this the value of λ changes dynamically during the simulation from 0 to 1 according to the function defined using the keyword *function* (described below), this switching from λ from 0 to 1 is done in t_s steps. Then comes the second equilibration period of t_{eq} to equilibrate the system in the $\lambda = 1$ state. After that, the switching back to the $\lambda = 0$ state is made using t_s timesteps and following the same switching function. After this period the value of λ is kept equal to zero and the fix has no other effect on the dynamics of the system.

The processes described above is known as nonequilibrium thermodynamic integration and is has been shown ([Freitas](#)) to present a much superior efficiency when compared to standard equilibrium methods. The reason why the switching it is made in both directions (potential to Einstein crystal and back) is to eliminate the dissipated heat due to the nonequilibrium process. Further details about nonequilibrium thermodynamic integration and its implementation in LAMMPS is available in [Freitas](#).

The *function* keyword allows the use of two different lambda paths. Option 1 results in a constant rate of change of λ with time:

$$\lambda(\tau) = \tau$$

where τ is the scaled time variable t/t_s . The option 2 performs the lambda switching at a rate defined by the following switching function

$$\lambda(\tau) = \tau^5 (70\tau^4 - 315\tau^3 + 540\tau^2 - 420\tau + 126)$$

This function has zero slope as λ approaches its extreme values (0 and 1), according to [de Koning](#) this results in smaller fluctuations on the integral to be computed on the thermodynamic integration. The use of option 2 is recommended since it results in better accuracy and less dissipation without any increase in computational resources cost.

Note

As described in [Freitas](#), it is important to keep the center-of-mass fixed during the thermodynamic integration. A nonzero total velocity will result in divergences during the integration due to the fact that the atoms are ‘attached’

to their equilibrium positions by the Einstein crystal. Check the option *zero* of [fix langevin](#) and [velocity](#). The use of the Nose-Hoover thermostat ([fix nvt](#)) is *NOT* recommended due to its well documented issues with the canonical sampling of harmonic degrees of freedom (notice that the *chain* option will *NOT* solve this problem). The Langevin thermostat ([fix langevin](#)) correctly thermostats the system and we advise its usage with *ti/spring* command.

2.236.4 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 global potential energy of the system as part of [thermodynamic output](#). The default setting for this fix is [fix_modify energy no](#).

This fix computes a global scalar and a global vector quantities 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 $0.5 \cdot k \cdot r^2$. The vector stores 2 values. The first value is the coupling parameter lambda. The second value is the derivative of lambda with respect to the integer timestep s, i.e. $\frac{d\lambda}{ds}$. In order to obtain $\frac{d\lambda}{dt}$, where t is simulation time, this 2nd value needs to be divided by the timestep size (e.g. 0.5 fs). 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.

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.

2.236.5 Related commands

[fix spring](#), [fix adapt](#)

2.236.6 Restrictions

This fix is part of the EXTRA-FIX package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

2.236.7 Default

The keyword default is function = 1.

(Freitas) Freitas, Asta, and de Koning, Computational Materials Science, 112, 333 (2016).

(de Koning) de Koning and Antonelli, Phys Rev E, 53, 465 (1996).

2.237 fix tmd command

2.237.1 Syntax

```
fix ID group-ID tmd rho_final file1 N file2
```

- ID, group-ID are documented in [fix](#) command
- tmd = style name of this fix command
- rho_final = desired value of rho at the end of the run (distance units)
- file1 = filename to read target structure from
- N = dump TMD statistics every this many timesteps, 0 = no dump
- file2 = filename to write TMD statistics to (only needed if N > 0)

2.237.2 Examples

```
fix 1 all nve
fix 2 tmdatoms tmd 1.0 target_file 100 tmd_dump_file
```

2.237.3 Description

Perform targeted molecular dynamics (TMD) on a group of atoms. A holonomic constraint is used to force the atoms to move towards (or away from) the target configuration. The parameter “rho” is monotonically decreased (or increased) from its initial value to rho_final at the end of the run.

Rho has distance units and is a measure of the root-mean-squared distance (RMSD) between the current configuration of the atoms in the group and the target coordinates listed in file1. Thus a value of rho_final = 0.0 means move the atoms all the way to the final structure during the course of the run.

The target file1 can be ASCII text or a gzipped text file (detected by a .gz suffix). The format of the target file1 is as follows:

```
0.0 25.0 xlo xhi
0.0 25.0 ylo yhi
0.0 25.0 zlo zhi
125    24.97311   1.69005   23.46956 0 0 -1
126    1.94691    2.79640   1.92799  1 0 0
127    0.15906    3.46099   0.79121  1 0 0
...
```

The first 3 lines may or may not be needed, depending on the format of the atoms to follow. If image flags are included with the atoms, the first 3 lo/hi lines **must** appear in the file. If image flags are not included, the first 3 lines **must not** appear. The 3 lines contain the simulation box dimensions for the atom coordinates, in the same format as in a LAMMPS data file (see the [read_data](#) command).

The remaining lines each contain an atom ID and its target x,y,z coordinates. The atom lines (all or none of them) can optionally be followed by 3 integer values: nx,ny,nz. For periodic dimensions, they specify which image of the box the atom is considered to be in, i.e. a value of N (positive or negative) means add N times the box length to the coordinate to get the true value. Those 3 integers either must be given for all atoms or none.

The atom lines can be listed in any order, but every atom in the group must be listed in the file. Atoms not in the fix group may also be listed; they will be ignored.

Comments starting with '#' and empty lines may be included as well.

TMD statistics are written to file2 every N timesteps, unless N is specified as 0, which means no statistics.

The atoms in the fix tmd group should be integrated (via a fix nve, nvt, npt) along with other atoms in the system.

Restarts can be used with a fix tmd command. For example, imagine a 10000 timestep run with a rho_initial = 11 and a rho_final = 1. If a restart file was written after 2000 time steps, then the configuration in the file would have a rho value of 9. A new 8000 time step run could be performed with the same rho_final = 1 to complete the conformational change at the same transition rate. Note that for restarted runs, the name of the TMD statistics file should be changed to prevent it being overwritten.

For more information about TMD, see ([Schlitter1](#)) and ([Schlitter2](#)).

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

No information about this fix is written to [binary restart files](#). None of the [fix_modify](#) options are relevant to this fix. No global or per-atom quantities are stored by this fix for access by various [output commands](#).

This fix can ramp its rho parameter over multiple runs, using the [start](#) and [stop](#) keywords of the [run](#) command. See the [run](#) command for details of how to do this.

This fix is not invoked during [energy minimization](#).

2.237.5 Restrictions

This fix is part of the EXTRA-FIX package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

All TMD fixes must be listed in the input script after all integrator fixes (nve, nvt, npt) are applied. This ensures that atoms are moved before their positions are corrected to comply with the constraint.

Atoms that have a TMD fix applied should not be part of a group to which a SHAKE fix is applied. This is because LAMMPS assumes there are not multiple competing holonomic constraints applied to the same atoms.

To read gzipped target files, you must compile LAMMPS with the -DLAMMPS_GZIP option. See the [Build settings](#) doc page for details.

2.237.6 Related commands

none

2.237.7 Default

none

(**Schlitter1**) Schlitter, Swegat, Mulders, “Distance-type reaction coordinates for modelling activated processes”, J Molecular Modeling, 7, 171-177 (2001).

(**Schlitter2**) Schlitter and Klahn, “The free energy of a reaction coordinate at multiple constraints: a concise formulation”, Molecular Physics, 101, 3439-3443 (2003).

2.238 fix ttm command

2.239 fix ttm/grid command

2.240 fix ttm/mod command

2.240.1 Syntax

```
fix ID group-ID ttm seed C_e rho_e kappa_e gamma_p gamma_s v_0 Nx Ny Nz keyword value ...
fix ID group-ID ttm/mod seed init_file Nx Ny Nz keyword value ...
```

- ID, group-ID are documented in [fix](#) command
 - style = *ttm* or *ttm/grid* or *ttm/mod*
 - seed = random number seed to use for white noise (positive integer)
 - remaining arguments for fix ttm or fix ttm/grid
- C_e = electronic specific heat (energy/(electron*temperature) units)
rho_e = electronic density (electrons/volume units)
kappa_e = electronic thermal conductivity (energy/(time*distance*temperature) units)
gamma_p = friction coefficient due to electron-ion interactions (mass/time units)
gamma_s = friction coefficient due to electronic stopping (mass/time units)
v_0 = electronic stopping critical velocity (velocity units)
Nx = number of thermal solve grid points in the x-direction (positive integer)
Ny = number of thermal solve grid points in the y-direction (positive integer)
Nz = number of thermal solve grid points in the z-direction (positive integer)

- remaining arguments for fix ttm/mod:

init_file = file with the parameters to TTM
Nx = number of thermal solve grid points in the x-direction (positive integer)
Ny = number of thermal solve grid points in the y-direction (positive integer)
Nz = number of thermal solve grid points in the z-direction (positive integer)

- zero or more keyword/value(s) pairs may be appended

- keyword = *set* or *infile* or *outfile*

set value = Tinit

Tinit = initial electronic temperature at all grid points (temperature units)

infile value = file.in with grid values for electronic temperatures

outfile values = Nout file.out

Nout = dump grid temperatures every this many timesteps

file.out = filename to write grid temperatures to

2.240.2 Examples

```
fix 2 all ttm 699489 1.0 1.0 10 0.1 0.0 2.0 1 12 1 infile initial outfile 1000 T.out
fix 3 all ttm/grid 123456 1.0 1.0 1.0 1.0 5.0 5 5 5 infile Te.in
fix 4 all ttm/mod 34277 parameters.txt 5 5 5 infile T_init outfile 10 T_out
```

Example input scripts using these commands can be found in examples/ttm.

2.240.3 Description

Use a two-temperature model (TTM) to represent heat transfer through and between electronic and atomic subsystems. LAMMPS models the atomic subsystem as usual with a molecular dynamics model and the classical force field specified by the user. The electronic subsystem is modeled as a continuum, or a background “gas”, on a regular grid which overlays the simulation domain. Energy can be transferred spatially within the grid representing the electrons. Energy can also be transferred between the electronic and atomic subsystems. The algorithm underlying this fix was derived by D. M. Duffy and A. M. Rutherford and is discussed in two J Physics: Condensed Matter papers: ([Duffy](#)) and ([Rutherford](#)). They used this algorithm in cascade simulations where a primary knock-on atom (PKA) was initialized with a high velocity to simulate a radiation event.

The description in this subsection applies to all 3 fix styles: *ttm*, *ttm/grid*, and *ttm/mod*.

Fix *ttm/grid* distributes the regular grid across processors consistent with the subdomains of atoms owned by each processor, but is otherwise identical to fix *ttm*. Note that fix *ttm* stores a copy of the grid on each processor, which is acceptable when the overall grid is reasonably small. For larger grids you should use fix *ttm/grid* instead.

Fix *ttm/mod* adds options to account for external heat sources (e.g. at a surface) and for specifying parameters that allow the electronic heat capacity to depend strongly on electronic temperature. It is more expensive computationally than fix *ttm* because it treats the thermal diffusion equation as non-linear. More details on fix *ttm/mod* are given below.

Heat transfer between the electronic and atomic subsystems is carried out via an inhomogeneous Langevin thermostat. Only atoms in the fix group contribute to and are affected by this heat transfer.

This thermostating differs from the regular Langevin thermostat ([fix langevin](#)) in three important ways. First, the Langevin thermostat is applied uniformly to all atoms in the user-specified group for a single target temperature, whereas the TTM fixes apply Langevin thermostating locally to atoms within the volumes represented by the user-specified grid points with a target temperature specific to that grid point. Second, the Langevin thermostat couples the temperature of the atoms to an infinite heat reservoir, whereas the heat reservoir for the TTM fixes is finite and represents the local electrons. Third, the TTM fixes allow users to specify not just one friction coefficient, but rather two independent friction coefficients: one for the electron-ion interactions (*gamma_p*), and one for electron stopping (*gamma_s*).

When the friction coefficient due to electron stopping, *gamma_s*, is non-zero, electron stopping effects are included for atoms moving faster than the electron stopping critical velocity, *v_0*. For further details about this algorithm, see ([Duffy](#)) and ([Rutherford](#)).

Energy transport within the electronic subsystem is solved according to the heat diffusion equation with added source terms for heat transfer between the subsystems:

$$C_e \rho_e \frac{\partial T_e}{\partial t} = \nabla(\kappa_e \nabla T_e) - g_p(T_e - T_a) + g_s T'_a$$

where C_e is the specific heat, ρ_e is the density, κ_e is the thermal conductivity, T is temperature, the “e” and “a” subscripts represent electronic and atomic subsystems respectively, g_p is the coupling constant for the electron-ion interaction, and g_s is the electron stopping coupling parameter. C_e , ρ_e , and κ_e are specified as parameters to the fix *ttm* or *ttm/grid*. The other quantities are derived. The form of the heat diffusion equation used here is almost the same as that in equation 6 of ([Duffy](#)), with the exception that the electronic density is explicitly represented, rather than being part of the specific heat parameter.

Currently, the TTM fixes assume that none of the user-supplied parameters will vary with temperature. Note that ([Duffy](#)) used a $\tanh()$ functional form for the temperature dependence of the electronic specific heat, but ignored temperature dependencies of any of the other parameters. See more discussion below for fix *ttm/mod*.

Note

These fixes do not perform time integration of the atoms in the fix group, they only rescale their velocities. Thus a time integration fix such as [fix nve](#) should be used in conjunction with these fixes. These fixes should not normally be used on atoms that have their temperature controlled by another thermostating fix, e.g. [fix nvt](#) or [fix langevin](#).

Note

These fixes require use of an orthogonal 3d simulation box with periodic boundary conditions in all dimensions. They also require that the size and shape of the simulation box do not vary dynamically, e.g. due to use of the [fix npt](#) command. Likewise, the size/shape of processor subdomains cannot vary due to dynamic load-balancing via use of the [fix balance](#) command. It is possible however to load balance before the simulation starts using the [balance](#) command, so that each processor has a different size subdomain.

Periodic boundary conditions are also used in the heat equation solve for the electronic subsystem. This varies from the approach of ([Rutherford](#)) where the atomic subsystem was embedded within a larger continuum representation of the electronic subsystem.

The *set* keyword specifies a *Tinit* temperature value to initialize the value stored on all grid points. By default the temperatures are all zero when the grid is created.

The *infile* keyword specifies an input file of electronic temperatures for each grid point to be read in to initialize the grid, as an alternative to using the *set* keyword.

The input file is a text file which may have comments starting with the '#' character. Each line contains four numeric columns: ix, iy, iz, Temperature. Empty or comment-only lines will be ignored. The number of lines must be equal to the number of user-specified grid points (Nx by Ny by Nz). The ix, iy, iz are grid point indices ranging from 1 to Nxzy inclusive in each dimension. The lines can appear in any order. For example, the initial electronic temperatures on a 1 by 2 by 3 grid could be specified in the file as follows:

```
# UNITS: metal COMMENT: initial electron temperature
1 1 1 1.0
1 1 2 1.0
1 1 3 1.0
1 2 1 2.0
1 2 2 2.0
1 2 3 2.0
```

where the electronic temperatures along the y=0 plane have been set to 1.0, and the electronic temperatures along the y=1 plane have been set to 2.0. If all the grid point values are not specified, LAMMPS will generate an error. LAMMPS will check if a "UNITS:" tag is in the first line and stop with an error, if there is a mismatch with the current units used.

Note

The electronic temperature at each grid point must be a non-zero positive value, both initially, and as the temperature evolves over time. Thus you must use either the *set* or *infile* keyword or be restarting a simulation that used this fix previously.

The *outfile* keyword has 2 values. The first value *Nout* triggers output of the electronic temperatures for each grid point every *Nout* timesteps. The second value is the filename for output, which will be suffixed by the timestep. The format of each output file is exactly the same as the input temperature file. It will contain a comment in the first line reporting the date the file was created, the LAMMPS units setting in use, grid size and the current timestep.

Note

The fix *ttm/grid* command does not support the *outfile* keyword. Instead you can use the *dump grid* command to output the electronic temperature on the distributed grid to a dump file or the *restart* command which creates a file specific to this fix which the *read restart* command reads. The file has the same format as the file the *infile* option reads.

For the fix *ttm* and fix *ttm/mod* commands, the corresponding atomic temperature for atoms in each grid cell can be computed and output by the *fix ave/chunk* command using the *compute chunk/atom* command to create a 3d array of chunks consistent with the grid used by this fix.

For the fix *ttm/grid* command the same thing can be done using the *fix ave/grid* command and its per-grid values can be output via the *dump grid* command.

Additional details for fix *ttm/mod*

Fix *ttm/mod* uses the heat diffusion equation with possible external heat sources (e.g. laser heating in ablation simulations):

$$C_e \rho_e \frac{\partial T_e}{\partial t} = \nabla(\kappa_e \nabla T_e) - g_p(T_e - T_a) + g_s T'_a + \theta(x - x_{surface}) I_0 \exp(-x/l_{skin})$$

where θ is the Heaviside step function, I_0 is the (absorbed) laser pulse intensity for ablation simulations, l_{skin} is the depth of the skin-layer, and all other designations have the same meaning as in the former equation. The duration of the pulse is set by the parameter *tau* in the *init_file*.

Fix *ttm/mod* also allows users to specify the dependencies of C_e and κ_e on the electronic temperature. The specific heat is expressed as

$$C_e = C_0 + (a_0 + a_1 X + a_2 X^2 + a_3 X^3 + a_4 X^4) \exp(-(AX)^2)$$

where $X = \frac{T_e}{1000}$, and the thermal conductivity is defined as $\kappa_e = D_e \cdot rho_e \cdot C_e$, where D_e is the thermal diffusion coefficient.

Electronic pressure effects are included in the TTM model to account for the blast force acting on ions because of electronic pressure gradient (see (*Chen*), (*Norman*))). The total force acting on an ion is:

$$\vec{F}_i = -\partial U / \partial \vec{r}_i + \vec{F}_{langevin} - \nabla P_e / n_{ion}$$

where $F_{langevin}$ is a force from Langevin thermostat simulating electron-phonon coupling, and $\nabla P_e / n_{ion}$ is the electron blast force.

The electronic pressure is taken to be $P_e = B \cdot rho_e \cdot C_e \cdot T_e$

The current fix *ttm/mod* implementation allows TTM simulations with a vacuum. The vacuum region is defined as the grid cells with zero electronic temperature. The numerical scheme does not allow energy exchange with such cells. Since the material can expand to previously unoccupied region in some simulations, the vacuum border can be allowed to move. It is controlled by the *surface_movement* parameter in the *init_file*. If it is set to 1, then “vacuum” cells can be changed to “electron-filled” cells with the temperature T_{e_min} if atoms move into them (currently only implemented for the case of 1-dimensional motion of a flat surface normal to the X axis). The initial locations of the interfaces of the

electron density to the vacuum can be set in the *init_file* via *lsurface* and *rsurface* parameters. In this case, electronic pressure gradient is calculated as

$$\nabla_x P_e = \left[\frac{C_e T_e(x) \lambda}{(x + \lambda)^2} + \frac{x}{x + \lambda} \frac{(C_e T_e)_{x+\Delta x} - (C_e T_e)_x}{\Delta x} \right]$$

where λ is the electron mean free path (see ([Norman](#)), ([Pisarev](#)))

The fix *ttm/mod* parameter file *init_file* has the following syntax. Every line with an odd number is considered as a comment and ignored. The lines with the even numbers are treated as follows:

```
a_0, energy/(temperature*electron) units
a_1, energy/(temperature^2*electron) units
a_2, energy/(temperature^3*electron) units
a_3, energy/(temperature^4*electron) units
a_4, energy/(temperature^5*electron) units
C_0, energy/(temperature*electron) units
A, 1/temperature units
rho_e, electrons/volume units
D_e, length^2/time units
gamma_p, mass/time units
gamma_s, mass/time units
v_0, length/time units
I_0, energy/(time*length^2) units
lsurface, electron grid units (positive integer)
rsurface, electron grid units (positive integer)
l_skin, length units
tau, time units
B, dimensionless
lambda, length units
n_ion, ions/volume units
surface_movement: 0 to disable tracking of surface motion, 1 to enable
T_e_min, temperature units
```

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

The fix ttm and fix ttm/mod commands write the state of the electronic subsystem and the energy exchange between the subsystems to *binary restart files*. The fix ttm/grid command does not yet support writing of its distributed grid to a restart file.

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. Note that the restart script must define the same size grid as the original script.

The fix ttm/grid command also outputs an auxiliary file each time a restart file is written, with the electron temperatures for each grid cell. The format of this file is the same as that read by the *infile* option explained above. The filename is the same as the restart filename with “.ttm” appended. This auxiliary file can be read in for a restarted run by using the *infile* option for the fix ttm/grid command, following the [read_restart](#) command.

None of the [fix_modify](#) options are relevant to these fixes.

These fixes compute 2 output quantities stored in a vector of length 2, which can be accessed by various *output commands*. The first quantity is the total energy of the electronic subsystem. The second quantity is the energy transferred from the electronic to the atomic subsystem on that timestep. Note that the velocity verlet integrator applies the fix ttm forces to the atomic subsystem as two half-step velocity updates: one on the current timestep and one on the subsequent

timestep. Consequently, the change in the atomic subsystem energy is lagged by half a timestep relative to the change in the electronic subsystem energy. As a result of this, users may notice slight fluctuations in the sum of the atomic and electronic subsystem energies reported at the end of the timestep.

The vector values calculated are “extensive”.

The fix ttm/grid command also outputs a per-grid vector which stores the electron temperature for each grid cell in temperature *units*, which can be accessed by various *output commands*. The length of the vector (distributed across all processors) is $Nx * Ny * Nz$. For access by other commands, the name of the single grid produced by fix ttm/grid is “grid”. The name of its per-grid data is “data”.

No parameter of the fixes can be used with the *start/stop* keywords of the *run* command. The fixes are not invoked during *energy minimization*.

2.240.5 Restrictions

All these fixes are part of the EXTRA-FIX package. They are only enabled if LAMMPS was built with that package. See the *Build package* page for more info.

As mentioned above, these fixes require 3d simulations and orthogonal simulation boxes periodic in all 3 dimensions.

These fixes used a random number generator to Langevin thermostat the electron temperature. This means you will not get identical answers when running on different numbers of processors or when restarting a simulation (even on the same number of processors). However, in a statistical sense, simulations on different processor counts and restarted simulation should produce results which are statistically the same.

2.240.6 Related commands

fix langevin, fix dt/reset

2.240.7 Default

none

(Duffy) D M Duffy and A M Rutherford, J. Phys.: Condens. Matter, 19, 016207-016218 (2007).

(Rutherford) A M Rutherford and D M Duffy, J. Phys.: Condens. Matter, 19, 496201-496210 (2007).

(Chen) J Chen, D Tzou and J Beraun, Int. J. Heat Mass Transfer, 49, 307-316 (2006).

(Norman) G E Norman, S V Starikov, V V Stegailov et al., Contrib. Plasma Phys., 53, 129-139 (2013).

(Pisarev) V V Pisarev and S V Starikov, J. Phys.: Condens. Matter, 26, 475401 (2014).

2.241 fix tune/kspace command

2.241.1 Syntax

```
fix ID group-ID tune/kspace N
```

- ID, group-ID are documented in *fix* command
- tune/kspace = style name of this fix command

- N = invoke this fix every N steps

2.241.2 Examples

```
fix 2 all tune/kspace 100
```

2.241.3 Description

This fix tests each kspace style (Ewald, PPPM, and MSM), and automatically selects the fastest style to use for the remainder of the run. If the fastest style is Ewald or PPPM, the fix also adjusts the Coulombic cutoff towards optimal speed. Future versions of this fix will automatically select other kspace parameters to use for maximum simulation speed. The kspace parameters may include the style, cutoff, grid points in each direction, order, Ewald parameter, MSM parallelization cut-point, MPI tasks to use, etc.

The rationale for this fix is to provide the user with as-fast-as-possible simulations that include long-range electrostatics (kspace) while meeting the user-prescribed accuracy requirement. A simple heuristic could never capture the optimal combination of parameters for every possible run-time scenario. But by performing short tests of various kspace parameter sets, this fix allows parameters to be tailored specifically to the user's machine, MPI ranks, use of threading or accelerators, the simulated system, and the simulation details. In addition, it is possible that parameters could be evolved with the simulation on-the-fly, which is useful for systems that are dynamically evolving (e.g. changes in box size/shape or number of particles).

When this fix is invoked, LAMMPS will perform short timed tests of various parameter sets to determine the optimal parameters. Tests are performed on-the-fly, with a new test initialized every N steps. N should be chosen large enough so that adequate CPU time lapses between tests, thereby providing statistically significant timings. But N should not be chosen to be so large that an unfortunate parameter set test takes an inordinate amount of wall time to complete. An N of 100 for most problems seems reasonable. Once an optimal parameter set is found, that set is used for the remainder of the run.

This fix uses heuristics to guide its selection of parameter sets to test, but the actual timed results will be used to decide which set to use in the simulation.

It is not necessary to discard trajectories produced using sub-optimal parameter sets, or a mix of various parameter sets, since the user-prescribed accuracy will have been maintained throughout. However, some users may prefer to use this fix only to discover the optimal parameter set for a given setup that can then be used on subsequent production runs.

This fix starts with kspace parameters that are set by the user with the [kspace_style](#) and [kspace_modify](#) commands. The prescribed accuracy will be maintained by this fix throughout the simulation.

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](#).

2.241.4 Restrictions

This fix is part of the KSPACE package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

Do not set “neigh_modify once yes” or else this fix will never be called. Renumbering is required.

This fix is not compatible with a hybrid pair style, long-range dispersion, TIP4P water support, or long-range point dipole support.

2.241.5 Related commands

kspace_style, boundary kspace_modify, pair_style lj/cut/coul/long, pair_style lj/charmm/coul/long, pair_style lj/long, pair_style lj/long/coul/long, pair_style buck/coul/long

2.241.6 Default

2.242 fix vector command

2.242.1 Syntax

```
fix ID group-ID vector Nevery value1 value2 ... keyword args ...
```

- ID, group-ID are documented in [fix](#) command
- vector = style name of this fix command
- Nevery = use input values 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 = value calculated by an equal-style variable with name
v_name[I] = Ith component of vector-style variable with name

- zero or more keyword/args pairs may be appended
 - keyword = *nmax*
- nmax length* = set maximal length of vector to <length>

2.242.2 Examples

```
fix 1 all vector 100 c_myTemp
fix 1 all vector 5 c_myTemp v_integral
fix 1 all vector 50 c_myTemp nmax 200
```

2.242.3 Description

Use one or more global values as inputs every few timesteps, and simply store them as a sequence. For a single specified value, the values are stored as a global vector of growing length. For multiple specified values, they are stored as rows in a global array, whose number of rows is growing. The resulting vector or array can be used by other [output commands](#).

The optional *nmax* keyword can be used to restrict the length of the vector to the given *length* value. Once the restricted vector is filled, the oldest entry will be discarded when a entry is added.

One way to use this command is to accumulate a vector that is numerically integrated using the [variable trap\(\)](#) function. For example, the velocity auto-correlation function (VACF) can be integrated, to yield a diffusion coefficient, as follows:

```
compute      2 all vacf
fix         5 all vector 1 c_2[4]
variable    diff equal dt*trap(f_5)
thermo_style custom step v_diff
```

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 or vector-style [variable](#). In each case, the compute, fix, or variable must produce a global quantity, not a per-atom or local quantity. And the global quantity must be a scalar, not a vector or array.

[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* or *vector* are the only ones that can be used with this fix. Variables of style *atom* cannot be used, since they produce per-atom values.

The *Nevery* argument specifies on what timesteps the input values will be used in order to be stored. Only timesteps that are a multiple of *Nevery*, including timestep 0, will contribute values.

Note

If *Nevery* is a small number and the simulation runs for many steps, the accumulated vector or array can become very large and thus consume a lot of memory. The implementation limit is about 2 billion entries. Using the *nmax* keyword mentioned above can avoid that by limiting the size of the vector.

Note that if you perform multiple runs, using the “pre no” option of the [run](#) command to avoid initialization on subsequent runs, then you need to use the [stop](#) keyword with the first [run](#) command with a timestep value that encompasses all the runs. This is so that the vector or array stored by this fix can be allocated to a sufficient size.

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 vector. Or it can be a compute defined not in your input script, but by [thermodynamic output](#) or other fixes such as [fix nvt](#) or [fix temp/rescale](#). See the doc pages for these commands which give the IDs of these computes. Users can also write code for their own compute styles and [add them to LAMMPS](#).

If a value begins with “f_”, a fix ID must follow which has been previously defined in the input script. If no bracketed term is appended, the global scalar calculated by the fix is used. If a bracketed term is appended, the Ith element of the global vector calculated by the fix is used.

Note that some fixes only produce their values on certain timesteps, which must be compatible with *Nevery*, else an error will result. Users can also write code for their own fix styles and [add them to LAMMPS](#).

If a value begins with “v_”, a variable name must follow which has been previously defined in the input script. An equal-style or vector-style variable can be referenced; the latter requires a bracketed term to specify the Ith element of the vector calculated by the variable. See the [variable](#) command for details. Note that variables of style *equal* and *vector* 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 be stored by fix vector.

2.242.4 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 or global array which can be accessed by various *output commands*. The values can only be accessed on timesteps that are multiples of *Nevery*.

A vector is produced if only a single input value is specified. An array is produced if multiple input values are specified. The length of the vector or the number of rows in the array grows by 1 every *Nevery* timesteps.

If the fix produces a vector, then the entire vector will be either “intensive” or “extensive”, depending on whether the values stored in the vector are “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 stored, then the compute or fix determines whether the value is intensive or extensive; see the page for that compute or fix for further info. Values produced by a variable are treated as intensive.

This fix can allocate storage for stored values accumulated over multiple runs, using the *start* and *stop* keywords of the *run* command. See the *run* command for details of how to do this. If using the *run pre no* command option, this is required to allow the fix to allocate sufficient storage for stored values.

This fix is not invoked during *energy minimization*.

2.242.5 Restrictions

none

2.242.6 Related commands

compute, *variable*

2.242.7 Defaults

The default value of *nmax* is deduced from the number of steps in a run (or multiple runs when using the *start* and *stop* keywords of the *run command*) divided by the choice of *Nevery* plus 1.

2.243 fix viscosity command

2.243.1 Syntax

```
fix ID group-ID viscosity N vdim pdim Nbin keyword value ...
```

- ID, group-ID are documented in *fix* command
- viscosity = style name of this fix command
- N = perform momentum exchange every N steps
- vdim = *x* or *y* or *z* = which momentum component to exchange
- pdim = *x* or *y* or *z* = direction of momentum transfer

- N_{bin} = # of layers in $pdim$ direction (must be even number)
 - zero or more keyword/value pairs may be appended
 - keyword = *swap* or *vtarget*
- swap value = N_{swap} = number of swaps to perform every N steps
vtarget value = V or INF = target velocity of swap partners (velocity units)

2.243.2 Examples

```
fix 1 all viscosity 100 x z 20
fix 1 all viscosity 50 x z 20 swap 2 vtargt 1.5
```

2.243.3 Description

Use the Muller-Plathe algorithm described in [this paper](#) to exchange momenta between two particles in different regions of the simulation box every N steps. This induces a shear velocity profile in the system. As described below this enables a viscosity of the fluid to be calculated. This algorithm is sometimes called a reverse non-equilibrium MD (reverse NEMD) approach to computing viscosity. This is because the usual NEMD approach is to impose a shear velocity profile on the system and measure the response via an off-diagonal component of the stress tensor, which is proportional to the momentum flux. In the Muller-Plathe method, the momentum flux is imposed, and the shear velocity profile is the system's response.

The simulation box is divided into N_{bin} layers in the $pdim$ direction, where the layer 1 is at the low end of that dimension and the layer N_{bin} is at the high end. Every N steps, N_{swap} pairs of atoms are chosen in the following manner. Only atoms in the fix group are considered. N_{swap} atoms in layer 1 with positive velocity components in the $vdim$ direction closest to the target value V are selected. Similarly, N_{swap} atoms in the “middle” layer (see below) with negative velocity components in the $vdim$ direction closest to the negative of the target value V are selected. The two sets of N_{swap} atoms are paired up and their $vdim$ momenta components are swapped within each pair. This resets their velocities, typically in opposite directions. Over time, this induces a shear velocity profile in the system which can be measured using commands such as the following, which writes the profile to the file `tmp.profile`:

```
compute layers all chunk/atom bin/1d z lower 0.05 units reduced
fix f1 all ave/chunk 100 10 1000 layers vx file tmp.profile
```

Note that by default, $N_{swap} = 1$ and *vtarget* = INF, though this can be changed by the optional *swap* and *vtarget* keywords. When *vtarget* = INF, one or more atoms with the most positive and negative velocity components are selected. Setting these parameters appropriately, in conjunction with the swap rate N , allows the momentum flux rate to be adjusted across a wide range of values, and the momenta to be exchanged in large chunks or more smoothly.

The “middle” layer for momenta swapping is defined as the $N_{bin}/2 + 1$ layer. Thus if $N_{bin} = 20$, the two swapping layers are 1 and 11. This should lead to a symmetric velocity profile since the two layers are separated by the same distance in both directions in a periodic sense. This is why N_{bin} is restricted to being an even number.

As described below, the total momentum transferred by these velocity swaps is computed by the fix and can be output. Dividing this quantity by time and the cross-sectional area of the simulation box yields a momentum flux. The ratio of momentum flux to the slope of the shear velocity profile is proportional to the viscosity of the fluid, in appropriate units. See the [Muller-Plathe paper](#) for details.

Note

If your system is periodic in the direction of the momentum flux, then the flux is going in 2 directions. This means the effective momentum flux in one direction is reduced by a factor of 2. You will see this in the equations for

viscosity in the Muller-Plathe paper. LAMMPS is simply tallying momentum which does not account for whether or not your system is periodic; you must use the value appropriately to yield a viscosity for your system.

Note

After equilibration, if the velocity profile you observe is not linear, then you are likely swapping momentum too frequently and are not in a regime of linear response. In this case you cannot accurately infer a viscosity and should try increasing the Nevery parameter.

An alternative method for calculating a viscosity is to run a NEMD simulation, as described on the [Howto nemd](#) doc page. NEMD simulations deform the simulation box via the [fix deform](#) command.

Some features or combination of settings in LAMMPS do not support non-orthogonal boxes. Using fix viscosity keeps the box orthogonal; thus it does not suffer from these limitations.

2.243.4 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 cumulative momentum transferred between the bottom and middle of the simulation box (in the *pdim* direction) is stored as a scalar quantity by this fix. This quantity is zeroed when the fix is defined and accumulates thereafter, once every N steps. The units of the quantity are momentum = mass*velocity. 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](#).

2.243.5 Restrictions

This fix is part of the EXTRA-FIX package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

Swaps conserve both momentum and kinetic energy, even if the masses of the swapped atoms are not equal. Thus you should not need to thermostat the system. If you do use a thermostat, you may want to apply it only to the non-swapped dimensions (other than *vdim*).

LAMMPS does not check, but you should not use this fix to swap velocities of atoms that are in constrained molecules, e.g. via [fix shake](#) or [fix rigid](#). This is because application of the constraints will alter the amount of transferred momentum. You should, however, be able to use flexible molecules. See the [Maginn paper](#) for an example of using this algorithm in a computation of alcohol molecule properties.

When running a simulation with large, massive particles or molecules in a background solvent, you may want to only exchange momenta between solvent particles.

2.243.6 Related commands

fix ave/chunk, fix thermal/conductivity

2.243.7 Default

The option defaults are swap = 1 and vtargt = INF.

(**Muller-Plathe**) Muller-Plathe, Phys Rev E, 59, 4894-4898 (1999).

(**Maginn**) Kelkar, Rafferty, Maginn, Siepmann, Fluid Phase Equilibria, 260, 218-231 (2007).

2.244 fix viscous command

Accelerator Variants: *viscous/kk*

2.244.1 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

2.244.2 Examples

```
fix 1 flow viscous 0.1
fix 1 damp viscous 0.5 scale 3 2.5
```

2.244.3 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 thermostating (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_i is given by $F_i = -\gamma v_i$. The larger the coefficient, the faster the kinetic energy is reduced. If the optional keyword *scale* is used, γ can scaled up or down by the specified factor for atoms of that type. It can be used multiple times to adjust γ for several atom types.

Note

You should specify gamma in force/velocity units. This is not the same as mass/time units, at least for some of the LAMMPS [units](#) options like “real” or “metal” that are not self-consistent.

In a Brownian dynamics context, $\gamma = \frac{k_B T}{D}$, where k_B = Boltzmann’s constant, T = temperature, and D = particle diffusion coefficient. D can be written as $\frac{k_B T}{3\pi\eta d}$, where η = 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, γ is specified directly in force/velocity units. If needed, γ can be adjusted for atoms of different sizes (i.e. σ) 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 square root of the chosen thermostating 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.

Styles with a *gpu*, *intel*, *kk*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed on the [Accelerator packages](#) page. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the GPU, INTEL, KOKKOS, OPENMP, and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the [Build package](#) page for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the [-suffix command-line switch](#) when you invoke LAMMPS, or you can use the [suffix](#) command in your input script.

See the [Accelerator packages](#) page for more instructions on how to use the accelerated styles effectively.

2.244.4 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 [fix_modify respa](#) option is supported by this fix. This allows to set at which level of the [r-RESPA](#) integrator the fix is modifying forces. Default is the outermost level.

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.

2.244.5 Restrictions

none

2.244.6 Related commands

fix langevin, fix viscous/sphere, fix damping/cundall

2.244.7 Default

none

2.245 fix viscous/sphere command

2.245.1 Syntax

```
fix ID group-ID viscous/sphere gamma keyword values ...
```

- ID, group-ID are documented in [fix](#) command
- viscous/sphere = style name of this fix command
- gamma = damping coefficient (torque/angular velocity units)
- zero or more keyword/value pairs may be appended

keyword = scale
 scale values = type ratio or v_name
 type = atom type (1-N)
 ratio = factor to scale the damping coefficients by
 v_name = reference to atom style variable name

2.245.2 Examples

```
fix 1 flow viscous/sphere 0.1
fix 1 damp viscous/sphere 0.5 scale 3 2.5
fix 1 damp viscous/sphere 0.5 scale v_radscale
```

2.245.3 Description

Add a viscous damping torque to finite-size spherical particles in the group that is proportional to the angular velocity of the atom. In granular simulations this can be useful for draining the rotational kinetic energy from the system in a controlled fashion. If used without additional thermostating (to add kinetic energy to the system), it has the effect of slowly (or rapidly) freezing the system; hence it can also be used as a simple energy minimization technique.

The damping torque T_i is given by $T_i = -\gamma\omega_i$. The larger the coefficient, the faster the rotational kinetic energy is reduced.

If the optional keyword *scale* is used, γ can be scaled up or down by the specified factor for atoms. This factor can be set for different atom types and thus the *scale* keyword used multiple times followed by the atom type and the

associated scale factor. Alternately the scaling factor can be computed for each atom (e.g. based on its radius) by using an *atom-style variable*.

Note

You should specify gamma in torque/angular velocity units. This is not the same as mass/time units, at least for some of the LAMMPS *units* options like “real” or “metal” that are not self-consistent.

In the current implementation, rather than have the user specify a viscosity, γ is specified directly in torque/angular velocity units. If needed, γ can be adjusted for atoms of different sizes (i.e. σ) by using the *scale* keyword.

2.245.4 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 *fix_modify respa* option is supported by this fix. This allows to set at which level of the *r-RESPA* integrator the fix is modifying torques. Default is the outermost level.

The torques 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.

2.245.5 Restrictions

This fix is part of the EXTRA-FIX package. It is only enabled if LAMMPS was built with that package. See the *Build package* page for more info.

This fix requires that atoms store torque and angular velocity (*omega*) and a radius as defined by the *atom_style sphere* command.

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

2.245.6 Related commands

fix viscous, fix damping/cundall

2.245.7 Default

none

2.246 fix wall/lj93 command

Accelerator Variants: *wall/lj93/kk*

2.247 fix wall/lj126 command

2.248 fix wall/lj1043 command

2.249 fix wall/colloid command

2.250 fix wall/harmonic command

2.251 fix wall/lepton command

2.252 fix wall/morse command

2.253 fix wall/table command

2.253.1 Syntax

```
fix ID group-ID style [tabstyle] [N] face args ... keyword value ...
```

- ID, group-ID are documented in [fix](#) command
- style = *wall/lj93* or *wall/lj126* or *wall/lj1043* or *wall/colloid* or *wall/harmonic* or *wall/lepton* or *wall/morse* or *wall/table*
- tabstyle = *linear* or *spline* = method of table interpolation (only applies to *wall/table*)
- N = use N values in *linear* or *spline* interpolation (only applies to *wall/table*)
- one or more face/arg pairs may be appended
- face = *xlo* or *xhi* or *ylo* or *yhi* or *zlo* or *zhi*
- args for styles *lj93* or *lj126* or *lj1043* or *colloid* or *harmonic*
 - args = coord epsilon sigma cutoff
 - coord = position of wall = EDGE or constant or variable
 - EDGE = current lo or hi edge of simulation box
 - constant = number like 0.0 or -30.0 (distance units)
 - variable = [equal-style](#) [variable](#) like v_x or v_wiggle
 - epsilon = strength factor for wall-particle interaction (energy or energy/distance^2 units)
 - epsilon can be a variable (see below)
 - sigma = size factor for wall-particle interaction (distance units)
 - sigma can be a variable (see below)
 - cutoff = distance from wall at which wall-particle interactions are cut off (distance units)
- args for style *lepton*

args = coord expression cutoff
coord = position of wall = EDGE or constant or variable
EDGE = current lo or hi edge of simulation box
constant = number like 0.0 or -30.0 (distance units)
variable = [equal-style variable](#) like v_x or v_wiggle
expression = Lepton expression for the potential (energy units)
cutoff = distance from wall at which wall-particle interactions are cut off (distance units)

- args for style *morse*

args = coord D_0 alpha r_0 cutoff
coord = position of wall = EDGE or constant or variable
EDGE = current lo or hi edge of simulation box
constant = number like 0.0 or -30.0 (distance units)
variable = [equal-style variable](#) like v_x or v_wiggle
D_0 = depth of the potential (energy units)
D_0 can be a variable (see below)
alpha = width factor for wall-particle interaction (1/distance units)
alpha can be a variable (see below)
r_0 = distance of the potential minimum from the face of region (distance units)
r_0 can be a variable (see below)
cutoff = distance from wall at which wall-particle interactions are cut off (distance units)

- args for style *table*

args = coord filename keyword cutoff
coord = position of wall = EDGE or constant or variable
EDGE = current lo or hi edge of simulation box
constant = number like 0.0 or -30.0 (distance units)
variable = [equal-style variable](#) like v_x or v_wiggle
filename = file containing tabulated energy and force values
keyword = section identifier to select a specific table in table file
cutoff = distance from wall at which wall-particle interactions are cut off (distance units)

- zero or more keyword/value pairs may be appended

- keyword = *units* or *fld* or *pbc*

units value = lattice or box
lattice = the wall position is defined in lattice units
box = the wall position is defined in simulation box units
fld value = yes or no
yes = invoke the wall constraint to be compatible with implicit FLD
no = invoke the wall constraint in the normal way
pbc value = yes or no
yes = allow periodic boundary in a wall dimension
no = require non-periodic boundaries in any wall dimension

2.253.2 Examples

```
fix wallhi all wall/lj93 xlo -1.0 1.0 1.0 2.5 units box
fix wallhi all wall/lj93 xhi EDGE 1.0 1.0 2.5
fix wallhi all wall/harmonic xhi EDGE 100.0 0.0 4.0 units box
fix wallhi all wall/morse xhi EDGE 1.0 1.0 1.0 2.5 units box
fix wallhi all wall/lj126 v_wiggle 23.2 1.0 1.0 2.5
fix zwalls all wall/colloid zlo 0.0 1.0 1.0 0.858 zhi 40.0 1.0 1.0 0.858
fix xwall mobile wall/table spline 200 EDGE -5.0 walltab.dat HARMONIC 4.0
fix xwalls mobile wall/lepton xlo -5.0 "k*(r-rc)^2;k=100.0" 4.0 xhi 5.0 "k*(r-rc)^2;k=100.0" 4.0
```

2.253.3 Description

Bound the simulation domain on one or more of its faces with a flat wall that interacts with the atoms in the group by generating a force on the atom in a direction perpendicular to the wall. The energy of wall-particle interactions depends on the style.

For style *wall/lj93*, the energy E is given by the 9-3 Lennard-Jones potential:

$$E = \epsilon \left[\frac{2}{15} \left(\frac{\sigma}{r} \right)^9 - \left(\frac{\sigma}{r} \right)^3 \right] \quad r < r_c$$

For style *wall/lj126*, the energy E is given by the 12-6 Lennard-Jones potential:

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

For style *wall/lj1043*, the energy E is given by the 10-4-3 Lennard-Jones potential:

$$E = 2\pi\epsilon \left[\frac{2}{5} \left(\frac{\sigma}{r} \right)^{10} - \left(\frac{\sigma}{r} \right)^4 - \frac{\sqrt{2}\sigma^3}{3(r + (0.61/\sqrt{2})\sigma)^3} \right] \quad r < r_c$$

For style *wall/colloid*, the energy E is given by an integrated form of the [pair_style colloid](#) potential:

$$E = \epsilon \left[\frac{\sigma^6}{7560} \left(\frac{6R - D}{D^7} + \frac{D + 8R}{(D + 2R)^7} \right) - \frac{1}{6} \left(\frac{2R(D + R) + D(D + 2R)[\ln D - \ln(D + 2R)]}{D(D + 2R)} \right) \right] \quad r < r_c$$

For style *wall/harmonic*, the energy E is given by a repulsive-only harmonic spring potential:

$$E = \epsilon (r - r_c)^2 \quad r < r_c$$

For style *wall/morse*, the energy E is given by a Morse potential:

$$E = D_0 [e^{-2\alpha(r-r_0)} - 2e^{-\alpha(r-r_0)}] \quad r < r_c$$

Added in version 28Mar2023.

For style *wall/lepton*, the energy E is provided as an Lepton expression string using “r” as the distance variable. The [Lepton library](#), that the *wall/lepton* style interfaces with, evaluates this expression string at run time to compute the wall-particle energy. It also creates an analytical representation of the first derivative of this expression with respect to “r” and then uses that to compute the force between the wall and atoms in the fix group. The Lepton expression must be either enclosed in quotes or must not contain any whitespace so that LAMMPS recognizes it as a single keyword.

Optionally, the expression may use “rc” to refer to the cutoff distance for the given wall. Further constants in the expression can be defined in the same string as additional expressions separated by semicolons. The expression “ $k^*(r_{rc})^2;k=100.0$ ” represents a repulsive-only harmonic spring as in fix *wall/harmonic* with a force constant K (same as ϵ above) of 100 energy units. More details on the Lepton expression strings are given below.

Added in version 28Mar2023.

For style *wall/table*, the energy E and forces are determined from interpolation tables listed in one or more files as a function of distance. The interpolation tables are used to evaluate energy and forces between particles and the wall similar to how analytic formulas are used for the other wall styles.

The interpolation tables are created as a pre-computation by fitting cubic splines to the file values and interpolating energy and force values at each of N distances. During a simulation, the tables are used to interpolate energy and force values as needed for each wall and particle separated by a distance R . The interpolation is done in one of two styles: *linear* or *spline*.

For the *linear* style, the distance R is used to find the 2 surrounding table values from which an energy or force is computed by linear interpolation.

For the *spline* style, cubic spline coefficients are computed and stored for each of the N values in the table, one set of splines for energy, another for force. Note that these splines are different than the ones used to pre-compute the N values. Those splines were fit to the *Nfile* values in the tabulated file, where often *Nfile* < N . The distance R is used to find the appropriate set of spline coefficients which are used to evaluate a cubic polynomial which computes the energy or force.

For each wall a filename and a keyword must be provided as in the examples above. The filename specifies a file containing tabulated energy and force values. The keyword specifies a section of the file. The format of this file is described below.

In all cases, r is the distance from the particle to the wall at position *coord*, and r_c is the *cutoff* distance at which the particle and wall no longer interact. The energy of the wall potential is shifted so that the wall-particle interaction energy is 0.0 at the cutoff distance.

Up to 6 walls or faces can be specified in a single command: *xlo, xhi, ylo, yhi, zlo, zhi*. A *lo* face interacts with particles near the lower side of the simulation box in that dimension. A *hi* face interacts with particles near the upper side of the simulation box in that dimension.

The position of each wall can be specified in one of 3 ways: as the EDGE of the simulation box, as a constant value, or as a variable. If EDGE is used, then the corresponding boundary of the current simulation box is used. If a numeric constant is specified then the wall is placed at that position in the appropriate dimension (x, y, or z). In both the EDGE and constant cases, the wall will never move. If the wall position is a variable, it should be specified as *v_name*, where *name* is an *equal-style variable* name. In this case the variable is evaluated each timestep and the result becomes the current position of the reflecting wall. Equal-style variables can specify formulas with various mathematical functions, and include *thermo_style* command keywords for the simulation box parameters and timestep and elapsed time. Thus it is easy to specify a time-dependent wall position. See examples below.

For the *wall/lj93* and *wall/lj126* and *wall/lj1043* styles, ϵ and σ 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 ϵ and σ may be different than any ϵ or σ values defined for a pair style that computes particle-particle interactions.

The *wall/lj93* interaction is derived by integrating over a 3d half-lattice of Lennard-Jones 12/6 particles. The *wall/lj126* interaction is effectively a harder, more repulsive wall interaction. The *wall/lj1043* interaction is yet a different form of wall interaction, described in Magda et al in ([Magda](#)).

For the *wall/colloid* style, 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 σ is the size of a constituent LJ particle inside the colloid particle and wall. Note that the cutoff distance R_c in this case is the distance from the colloid particle center to the wall. The prefactor ϵ can be thought of as an effective Hamaker constant with energy units for the strength of the colloid-wall interaction. More specifically, the ϵ prefactor is $4\pi^2 \rho_{wall} \rho_{colloid} \epsilon \sigma^6$, where ϵ and σ are the LJ parameters for the constituent LJ particles. ρ_{wall} and $\rho_{colloid}$ are the number density of the constituent particles, in the wall and colloid respectively, in units of 1/volume.

The *wall/colloid* interaction is derived by integrating over constituent LJ particles of size σ within the colloid particle and a 3d half-lattice of Lennard-Jones 12/6 particles of size σ in the wall. As mentioned in the preceding paragraph, the density of particles in the wall and colloid can be different, as specified by the ϵ prefactor.

For the *wall/harmonic* style, ϵ is effectively the spring constant K, and has units (energy/distance²). The input parameter σ 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*.

For the *wall/morse* style, the three parameters are in this order: D_0 the depth of the potential, α the width parameter, and r_0 the location of the minimum. D_0 has energy units, α inverse distance units, and r_0 distance units.

For any wall that supports them, the ϵ and/or σ and/or α parameter can be specified as an *equal-style variable*, in which case it should be specified as v_name, where name is the variable name. As with a variable wall position, the variable is evaluated each timestep and the result becomes the current epsilon or sigma of the 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 interaction.

Note

For all of the styles, you must ensure that r is always > 0 for all particles in the group, or LAMMPS will generate an error. This means you cannot start your simulation with particles at the wall position *coord* ($r = 0$) or with particles on the wrong side of the wall ($r < 0$). For the *wall/lj93* and *wall/lj126* styles, the energy of the wall/particle interaction (and hence the force on the particle) blows up as $r \rightarrow 0$. The *wall/colloid* style is even more restrictive, since the energy blows up as $D = r - R \rightarrow 0$. This means the finite-size particles of radius R must be a distance larger than R from the wall position *coord*. The *harmonic* style is a softer potential and does not blow up as $r \rightarrow 0$, but you must use a large enough ϵ that particles always remain on the correct side of the wall ($r > 0$).

The *units* keyword determines the meaning of the distance units used to define a wall position, but only when a numeric constant 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.

The *fld* keyword can be used with a *yes* setting to invoke the wall constraint before pairwise interactions are computed. This allows an implicit FLD model using *pair_style lubricateU* to include the wall force in its calculations. If the setting is *no*, wall forces are imposed after pairwise interactions, in the usual manner.

The *pbc* keyword can be used with a *yes* setting to allow walls to be specified in a periodic dimension. See the *boundary* command for options on simulation box boundaries. The default for *pbc* is *no*, which means the system must be non-periodic when using a wall. But you may wish to use a periodic box. E.g. to allow some particles to interact with the wall via the fix group-ID, and others to pass through it and wrap around a periodic box. In this case you should ensure that the wall is sufficiently far enough away from the box boundary. If you do not, then particles may interact with both the wall and with periodic images on the other side of the box, which is probably not what you want.

Here are examples of variable definitions that move the wall position in a time-dependent fashion using equal-style *variables*. The wall interaction parameters (epsilon, sigma) could be varied with additional variable definitions.

```
variable ramp equal ramp(0,10)
fix 1 all wall xlo v_ramp 1.0 1.0 2.5
```

```
variable linear equal vdisplace(0,20)
fix 1 all wall xlo v_linear 1.0 1.0 2.5
```

(continues on next page)

(continued from previous page)

```
variable wiggle equal swiggle(0.0,5.0,3.0)
fix 1 all wall xlo v_wiggle 1.0 1.0 2.5
```

```
variable wiggle equal cwiggle(0.0,5.0,3.0)
fix 1 all wall xlo v_wiggle 1.0 1.0 2.5
```

The *ramp(*lo,hi*)* function adjusts the wall position linearly from *lo* to *hi* over the course of a run. The *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*:

$$\text{position} = \text{c0} + \text{A} \sin(\text{omega} * \text{delta})$$

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:

$$\text{position} = \text{c0} + \text{A} (1 - \cos(\text{omega} * \text{delta}))$$

2.253.4 Lepton expression syntax and features

Lepton supports the following operators in expressions:

+	Add	-	Subtract	*	Multiply	/	Divide	^	Power
----------	-----	----------	----------	----------	----------	----------	--------	----------	-------

The following mathematical functions are available:

<code>sqrt(x)</code>	Square root	<code>exp(x)</code>	Exponential
<code>log(x)</code>	Natural logarithm	<code>sin(x)</code>	Sine (angle in radians)
<code>cos(x)</code>	Cosine (angle in radians)	<code>sec(x)</code>	Secant (angle in radians)
<code>csc(x)</code>	Cosecant (angle in radians)	<code>tan(x)</code>	Tangent (angle in radians)
<code>cot(x)</code>	Cotangent (angle in radians)	<code>asin(x)</code>	Inverse sine (in radians)
<code>acos(x)</code>	Inverse cosine (in radians)	<code>atan(x)</code>	Inverse tangent (in radians)
<code>sinh(x)</code>	Hyperbolic sine	<code>cosh(x)</code>	Hyperbolic cosine
<code>tanh(x)</code>	Hyperbolic tangent	<code>erf(x)</code>	Error function
<code>erfc(x)</code>	Complementary Error function	<code>abs(x)</code>	Absolute value
<code>min(x,y)</code>	Minimum of two values	<code>max(x,y)</code>	Maximum of two values
<code>delta(x)</code>	<code>delta(x)</code> is 1 for <i>x</i> = 0, otherwise 0	<code>step(x)</code>	<code>step(x)</code> is 0 for <i>x</i> < 0, otherwise 1

Numbers may be given in either decimal or exponential form. All of the following are valid numbers: *5*, *-3.1*, *1e6*, and *3.12e-2*.

As an extension to the standard Lepton syntax, it is also possible to use LAMMPS *variables* in the format “v_name”. Before evaluating the expression, “v_name” will be replaced with the value of the variable “name”. This is compatible with all kinds of scalar variables, but not with vectors, arrays, local, or per-atom variables. If necessary, a custom scalar variable needs to be defined that can access the desired (single) item from a non-scalar variable. As an example, the following lines will instruct LAMMPS to ramp the force constant for a harmonic bond from 100.0 to 200.0 during the next run:

```
variable fconst equal ramp(100.0, 200)
bond_style lepton
bond_coeff 1 1.5 "v_fconst * (r^2)"
```

An expression may be followed by definitions for intermediate values that appear in the expression. A semicolon “;” is used as a delimiter between value definitions. For example, the expression:

```
a^2+a*b+b^2; a=a1+a2; b=b1+b2
```

is exactly equivalent to

```
(a1+a2)^2+(a1+a2)*(b1+b2)+(b1+b2)^2
```

The definition of an intermediate value may itself involve other intermediate values. Whitespace and quotation characters (‘’ and “”) are ignored. All uses of a value must appear *before* that value’s definition. For efficiency reasons, the expression string is parsed, optimized, and then stored in an internal, pre-parsed representation for evaluation.

Evaluating a Lepton expression is typically between 2.5 and 5 times slower than the corresponding compiled and optimized C++ code. If additional speed or GPU acceleration (via GPU or KOKKOS) is required, the interaction can be represented as a table. Suitable table files can be created either internally using the [pair_write](#) or [bond_write](#) command or through the Python scripts in the [tools/tabulate](#) folder.

2.253.5 Table file format

Suitable tables for use with fix *wall/table* can be created by the Python code in the tools/tabulate folder of the LAMMPS source code distribution.

The format of a tabulated file is as follows (without the parenthesized comments):

```
# Tabulated wall potential UNITS: real

HARMONIC           (keyword is the first text on a line)
N 100 FP 200 200
                    (blank line)
1 0.04 1568.16 792.00 (index, distance to wall, energy, force)
2 0.08 1536.64 784.00
3 0.12 1505.44 776.00
...
99 3.96    0.16   8.00
100 4.00   0       0
```

A section begins with a non-blank line whose first character is not a “#”; blank lines or lines starting with “#” can be used as comments between sections. The first line begins with a keyword which identifies the section. The line can contain additional text, but the initial text must match the argument specified in the fix *wall/table* command. The next line lists (in any order) one or more parameters for the table. Each parameter is a keyword followed by one or more numeric values.

The parameter “N” is required and its value is the number of table entries that follow. Note that this may be different than the *N* specified in the fix *wall/table* command. Let *Ntable* = *N* in the fix command, and *Nfile* = “*N*” in the tabulated file. What LAMMPS does is a preliminary interpolation by creating splines using the *Nfile* tabulated values as nodal points. It uses these to interpolate as needed to generate energy and force values at *Ntable* different points. The resulting tables of length *Ntable* are then used as described above, when computing energy and force for wall-particle interactions.

This means that if you want the interpolation tables of length Ntable to match exactly what is in the tabulated file (with effectively no preliminary interpolation), you should set Ntable = Nfile.

2.253.6 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 all the specified walls to the global potential energy of the system as part of *thermodynamic output*. The default setting for this fix is *fix_modify energy no*.

The *fix_modify virial* option is supported by this fix to add the contribution due to the interaction between atoms and all the specified walls to both the global pressure and per-atom stress of the system via the *compute pressure* and *compute stress/atom* commands. The former can be accessed by *thermodynamic output*. The default setting for this fix is *fix_modify virial no*.

The *fix_modify respa* option is supported by this fix. This allows to set at which level of the *r-RESPA* integrator the fix is adding its forces. Default is the outermost level.

This fix computes a global scalar energy and a global vector of forces, which can be accessed by various *output commands*. Note that the scalar energy is the sum of interactions with all defined walls. If you want the energy on a per-wall basis, you need to use multiple fix wall commands. The length of the vector is equal to the number of walls defined by the fix. Each vector value is the normal force on a specific wall. Note that an outward force on a wall will be a negative value for *lo* walls and a positive value for *hi* walls. The scalar and vector values calculated by this fix are “extensive”.

No parameter of this fix can be used with the *start/stop* keywords of the *run* command.

The forces due to this fix are imposed during an energy minimization, invoked by the *minimize* command.

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.

Styles with a *gpu*, *intel*, *kk*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed on the *Accelerator packages* page. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the GPU, INTEL, KOKKOS, OPENMP, and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the *Build package* page for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the *-suffix command-line switch* when you invoke LAMMPS, or you can use the *suffix* command in your input script.

See the *Accelerator packages* page for more instructions on how to use the accelerated styles effectively.

2.253.7 Restrictions

Fix *wall/lepton* is part of the LEPTON package and only enabled if LAMMPS was built with this package. See the [Build package](#) page for more info.

2.253.8 Related commands

fix wall/reflect, *fix wall/gran*, *fix wall/region*

2.253.9 Default

The option defaults units = lattice, fld = no, and pbc = no.

(**Magda**) Magda, Tirrell, Davis, J Chem Phys, 83, 1888-1901 (1985); erratum in JCP 84, 2901 (1986).

2.254 fix wall/body/polygon command

2.254.1 Syntax

`fix ID group-ID wall/body/polygon k_n c_n c_t wallstyle args keyword values ...`

- ID, group-ID are documented in [fix](#) command
- wall/body/polygon = style name of this fix command
- k_n = normal repulsion strength (force/distance or pressure units)
- c_n = normal damping coefficient (force/distance or pressure units)
- c_t = tangential damping coefficient (force/distance or pressure units)
- wallstyle = *xplane* or *yplane* or *zcylinder*
- args = list of arguments for a particular style

xplane or *yplane* args = lo hi

lo,hi = position of lower and upper plane (distance units), either can be NULL

zcylinder args = radius

radius = cylinder radius (distance units)

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

- keyword = *wiggle*

wiggle values = dim amplitude period

dim = x or y or z

amplitude = size of oscillation (distance units)

period = time of oscillation (time units)

2.254.2 Examples

```
fix 1 all wall/body/polygon 1000.0 20.0 5.0 xplane -10.0 10.0
```

2.254.3 Description

This fix is for use with 2d models of body particles of style *rounded/polygon*. It bounds the simulation domain with wall(s). All particles in the group interact with the wall when they are close enough to touch it. The nature of the interaction between the wall and the polygon particles is the same as that between the polygon particles themselves, which is similar to a Hookean potential. See the [Howto body](#) page for more details on using body particles.

The parameters *k_n*, *c_n*, *c_t* have the same meaning and units as those specified with the [pair_style body/rounded/polygon](#) command.

The *wallstyle* can be planar or cylindrical. The 2 planar options specify a pair of walls in a dimension. Wall positions are given by *lo* and *hi*. Either of the values can be specified as NULL if a single wall is desired. For a *zcylinder* wallstyle, the cylinder's axis is at *x = y = 0.0*, and the radius of the cylinder is specified.

Optionally, the wall can be moving, if the *wiggle* keyword is appended.

For the *wiggle* keyword, the wall oscillates sinusoidally, similar to the oscillations of particles which can be specified by the [fix move](#) command. This is useful in packing simulations of particles. The arguments to the *wiggle* keyword specify a dimension for the motion, as well as its *amplitude* and *period*. Note that if the dimension is in the plane of the wall, this is effectively a shearing motion. If the dimension is perpendicular to the wall, it is more of a shaking motion. A *zcylinder* wall can only be wiggled in the z dimension.

Each timestep, the position of a wiggled wall in the appropriate *dim* is set according to this equation:

position = *coord* + *A* - *A* cos (*omega* * *delta*)

where *coord* is the specified initial position of the wall, *A* is the *amplitude*, *omega* is $2\pi / \text{period}$, and *delta* is the time elapsed since the fix was specified. The velocity of the wall is set to the derivative of this expression.

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

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*.

2.254.5 Restrictions

This fix is part of the BODY package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

Any dimension (xy) that has a wall must be non-periodic.

2.254.6 Related commands

atom_style body, pair_style body/rounded/polygon

2.254.7 Default

none

2.255 fix wall/body/polyhedron command

2.255.1 Syntax

```
fix ID group-ID wall/body/polyhedron k_n c_n c_t wallstyle args keyword values ...
```

- ID, group-ID are documented in [fix](#) command
- wall/body/polyhedron = style name of this fix command
- k_n = normal repulsion strength (force/distance units or pressure units - see discussion below)
- c_n = normal damping coefficient (force/distance units or pressure units - see discussion below)
- c_t = tangential damping coefficient (force/distance units or pressure units - see discussion below)
- wallstyle = *xplane* or *yplane* or *zplane*
- args = list of arguments for a particular style
 - xplane or yplane or zplane args = lo hi
lo,hi = position of lower and upper plane (distance units), either can be NULL)
- zero or more keyword/value pairs may be appended to args
- keyword = *wiggle*
 - wiggle values = dim amplitude period
 - dim = x or y or z
 - amplitude = size of oscillation (distance units)
 - period = time of oscillation (time units)

2.255.2 Examples

```
fix 1 all wall/body/polyhedron 1000.0 20.0 5.0 xplane -10.0 10.0
```

2.255.3 Description

This fix is for use with 3d models of body particles of style *rounded/polyhedron*. It bounds the simulation domain with wall(s). All particles in the group interact with the wall when they are close enough to touch it. The nature of the interaction between the wall and the polygon particles is the same as that between the polygon particles themselves, which is similar to a Hookean potential. See the [Howto body](#) page for more details on using body particles.

The parameters k_n , c_n , c_t have the same meaning and units as those specified with the [pair_style body/rounded/polyhedron](#) command.

The *wallstyle* can be planar or cylindrical. The 3 planar options specify a pair of walls in a dimension. Wall positions are given by *lo* and *hi*. Either of the values can be specified as NULL if a single wall is desired.

Optionally, the wall can be moving, if the *wiggle* keyword is appended.

For the *wiggle* keyword, the wall oscillates sinusoidally, similar to the oscillations of particles which can be specified by the [fix move](#) command. This is useful in packing simulations of particles. The arguments to the *wiggle* keyword specify a dimension for the motion, as well as its *amplitude* and *period*. Note that if the dimension is in the plane of the wall, this is effectively a shearing motion. If the dimension is perpendicular to the wall, it is more of a shaking motion.

Each timestep, the position of a wiggled wall in the appropriate *dim* is set according to this equation:

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

where *coord* is the specified initial position of the wall, *A* is the *amplitude*, *omega* is $2\pi / \text{period}$, and *delta* is the time elapsed since the fix was specified. The velocity of the wall is set to the derivative of this expression.

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

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](#).

2.255.5 Restrictions

This fix is part of the BODY package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

Any dimension (xyz) that has a wall must be non-periodic.

2.255.6 Related commands

atom_style body, *pair_style body/rounded/polyhedron*

2.255.7 Default

none

2.256 fix wall/ees command

2.257 fix wall/region/ees command

2.257.1 Syntax

`fix ID group-ID style args`

- ID, group-ID are documented in [fix](#) command
- style = *wall/ees* or *wall/region/ees*

args for style *wall/ees*: one or more face parameters groups may be appended
 face = xlo or xhi or ylo or yhi or zlo or zhi

parameters = coord epsilon sigma cutoff

coord = position of wall = EDGE or constant or variable

EDGE = current lo or hi edge of simulation box

constant = number like 0.0 or -30.0 (distance units)

variable = [equal-style variable](#) like v_x or v_wiggle

epsilon = strength factor for wall-particle interaction (energy or energy/distance² units)

epsilon can be a variable (see below)

sigma = size factor for wall-particle interaction (distance units)

sigma can be a variable (see below)

cutoff = distance from wall at which wall-particle interaction is cut off (distance units)

args for style *wall/region/ees*: region-ID epsilon sigma cutoff

region-ID = region whose boundary will act as wall

epsilon = strength factor for wall-particle interaction (energy or energy/distance² units)

sigma = size factor for wall-particle interaction (distance units)

cutoff = distance from wall at which wall-particle interaction is cut off (distance units)

2.257.2 Examples

```
fix wallhi all wall/ees xlo -1.0 1.0 1.0 2.5 units box
fix wallhi all wall/ees xhi EDGE 1.0 1.0 2.5
fix wallhi all wall/ees v_wiggle 23.2 1.0 1.0 2.5
fix zwalls all wall/ees zlo 0.0 1.0 1.0 0.858 zhi 40.0 1.0 1.0 0.858

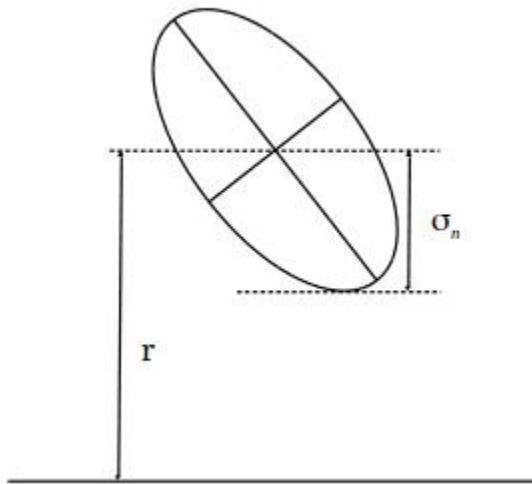
fix ees_cube all wall/region/ees myCube 1.0 1.0 2.5
```

2.257.3 Description

Fix `wall/ees` bounds the simulation domain on one or more of its faces with a flat wall that interacts with the ellipsoidal atoms in the group by generating a force on the atom in a direction perpendicular to the wall and a torque parallel with the wall. The energy of wall-particle interactions E is given by:

$$E = \epsilon \left[\frac{2\sigma_{LJ}^{12} (7r^5 + 14r^3\sigma_n^2 + 3r\sigma_n^4)}{945 (r^2 - \sigma_n^2)^7} - \frac{\sigma_{LJ}^6 (2r\sigma_n^3 + \sigma_n^2(r^2 - \sigma_n^2) \log \left[\frac{r - \sigma_n}{r + \sigma_n} \right])}{12\sigma_n^5 (r^2 - \sigma_n^2)} \right] \quad \sigma_n < r < r_c$$

Introduced by Babadi and Ejtehadi in ([Babadi2](#)). Here, r is the distance from the particle to the wall at position `coord`, and R_c is the *cutoff* distance at which the particle and wall no longer interact. Also, σ_n is the distance between center of ellipsoid and the nearest point of its surface to the wall as shown below.



Details of using this command and specifications are the same as fix/wall command. You can also find an example in `USER/ees/` under examples/ directory.

The prefactor ϵ can be thought of as an effective Hamaker constant with energy units for the strength of the ellipsoid-wall interaction. More specifically, the ϵ prefactor is

$$8\pi^2 \rho_{wall} \rho_{ellipsoid} \epsilon \sigma_a \sigma_b \sigma_c$$

where ϵ is the LJ energy parameter for the constituent LJ particles and σ_a , σ_b , and σ_c are the radii of the ellipsoidal particles. ρ_{wall} and $\rho_{ellipsoid}$ are the number density of the constituent particles, in the wall and ellipsoid respectively, in units of 1/volume.

Note

You must ensure that r is always bigger than σ_n for all particles in the group, or LAMMPS will generate an error. This means you cannot start your simulation with particles touching the wall position `coord` ($r = \sigma_n$) or with particles penetrating the wall ($0 \leq r < \sigma_n$) or with particles on the wrong side of the wall ($r < 0$).

Fix `wall/region/ees` treats the surface of the geometric region defined by the `region-ID` as a bounding wall which interacts with nearby ellipsoidal particles according to the EES potential introduced above.

Other details of this command are the same as for the [fix wall/region](#) command. One may also find an example of using this fix in the examples/PACKAGES/ees/ directory.

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

No information about these fixes are written to [binary restart files](#).

The [fix_modify energy](#) option is supported by these fixes to add the energy of interaction between atoms and all the specified walls or region wall to the global potential energy of the system as part of [thermodynamic output](#). The default settings for these fixes are [fix_modify energy no](#).

The [fix_modify respa](#) option is supported by these fixes. This allows to set at which level of the [r-RESPA](#) integrator the fix is adding its forces. Default is the outermost level.

These fixes computes a global scalar and a global vector of forces, which can be accessed by various [output commands](#). See the [fix wall](#) command for a description of the scalar and vector.

No parameter of these fixes can be used with the [start/stop](#) keywords of the [run](#) command.

The forces due to these fixes are imposed during an energy minimization, invoked by the [minimize](#) command.

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.

2.257.5 Restrictions

These fixes are part of the EXTRA-FIX package. They are only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

These fixes requires that atoms be ellipsoids as defined by the [atom_style ellipsoid](#) command.

2.257.6 Related commands

[fix wall](#), [pair resquared](#)

2.257.7 Default

none

(**Babadi2**) Babadi and Ejtehadi, EPL, 77 (2007) 23002.

2.258 fix wall/flow command

Accelerator Variants: *wall/flow/kk*

2.258.1 Syntax

```
fix ID group-ID wall/flow axis vflow T seed N coords ... keyword value
```

- ID, group-ID are documented in [fix](#) command
- wall/flow = style name of this fix command
- axis = flow axis (*x*, *y*, or *z*)
- vflow = generated flow velocity in *axis* direction (velocity units)
- T = flow temperature (temperature units)
- seed = random seed for stochasticity (positive integer)
- N = number of walls
- coords = list of N wall positions along the *axis* direction in ascending order (distance units)
- zero or more keyword/value pairs may be appended
- keyword = *units*
units value = lattice or box
lattice = wall positions are defined in lattice units
box = the wall positions are defined in simulation box units

2.258.2 Examples

```
fix 1 all wall/flow x 0.4 1.5 593894 4 2.0 4.0 6.0 8.0
```

2.258.3 Description

Added in version 17Apr2024.

This fix implements flow boundary conditions (FBC) introduced in ([Pavlov1](#)) and ([Pavlov2](#)). The goal is to generate a stationary flow with a shifted Maxwell velocity distribution:

$$f_a(v_a) \propto \exp\left(-\frac{m(v_a - v_{\text{flow}})^2}{2kBT}\right)$$

where v_a is the component of velocity along the specified *axis* argument ($a = x, y, z$), v_{flow} is the flow velocity specified as the *vflow* argument, T is the specified flow temperature, m is the particle mass, and kB is the Boltzmann constant.

This is achieved by defining a series of N transparent walls along the flow *axis* direction. Each wall is at the specified position listed in the *coords* argument. Note that an additional transparent wall is defined by the code at the boundary of the (periodic) simulation domain in the *axis* direction. So there are effectively $N+1$ walls.

Each time a particle in the specified group passes through one of the transparent walls, its velocity is re-assigned. Particles not in the group do not interact with the wall. This can be used, for example, to add obstacles composed of atoms, or to simulate a solution of complex molecules in a one-atom liquid (note that the fix has been tested for one-atom systems only).

Conceptually, the velocity re-assignment represents creation of a new particle within the system with simultaneous removal of the particle which passed through the wall. The velocity components in directions parallel to the wall are re-assigned according to the standard Maxwell velocity distribution for the specified temperature T . The velocity component perpendicular to the wall is re-assigned according to the shifted Maxwell distribution defined above:

$$f_{\text{a generated}}(v_a) \propto v_a f_a(v_a)$$

It can be shown that for an ideal-gas scenario this procedure makes the velocity distribution of particles between walls exactly as desired.

Since in most cases simulated systems are not an ideal gas, multiple walls can be defined, since a single wall may not be sufficient for maintaining a stationary flow without “congestion” which can manifest itself as regions in the flow with increased particle density located upstream from static obstacles.

For the same reason, the actual temperature and velocity of the generated flow may differ from what is requested. The degree of discrepancy is determined by how different from an ideal gas the simulated system is. Therefore, a calibration procedure may be required for such a system as described in ([Pavlov](#)).

Note that the interactions between particles on different sides of a transparent wall are not disabled or neglected. Likewise particle positions are not altered by the velocity reassignment. This removes the need to modify the force field to work correctly in cases when a particle is close to a wall.

For example, if particle positions were uniformly redistributed across the surface of a wall, two particles could end up too close to each other, potentially causing the simulation to explode. However due to this compromise, some collective phenomena such as regions with increased/decreased density or collective movements are not fully removed when particles cross a wall. This unwanted consequence can also be potentially mitigated by using more multiple walls.

Note

When the specified flow has a high velocity, a lost atoms error can occur (see [error messages](#)). If this happens, you should ensure the checks for neighbor list rebuilds, set via the `neigh_modify` command, are as conservative as possible (every timestep if needed). Those are the default settings.

2.258.4 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](#).

2.258.5 Restrictions

Fix `wall_flow` is part of the EXTRA-FIX package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

Flow boundary conditions should not be used with rigid bodies such as those defined by a “fix rigid” command.

This fix can only be used with periodic boundary conditions along the flow axis. The size of the box in this direction must not change. Also, the fix is designed to work only in an orthogonal simulation box.

2.258.6 Related commands

fix wall/reflect command

2.258.7 Default

The default for the units keyword is lattice.

(Pavlov1) Pavlov, Kolotinskii, Stegailov, “GPU-Based Molecular Dynamics of Turbulent Liquid Flows with OpenMM”, Proceedings of PPAM-2022, LNCS (Springer), vol. 13826, pp. 346-358 (2023)

(Pavlov2) Pavlov, Galigerov, Kolotinskii, Nikolskiy, Stegailov, “GPU-based Molecular Dynamics of Fluid Flows: Reaching for Turbulence”, Int. J. High Perf. Comp. Appl., (2024)

2.259 fix wall/gran command

Accelerator Variants: *wall/gran/kk*

2.259.1 Syntax

```
fix ID group-ID wall/gran fstyle fstyle_params wallstyle args keyword values ...
```

- ID, group-ID are documented in [fix](#) command
- wall/gran = style name of this fix command
- fstyle = style of force interactions between particles and wall

possible choices: hooke, hooke/history, hertz/history, granular

- fstyle_params = parameters associated with force interaction style

For hooke, hooke/history, and hertz/history, fstyle_params are:

Kn = elastic constant for normal particle repulsion (force/distance units or pressure units - see [discussion below](#))

Kt = elastic constant for tangential contact (force/distance units or pressure units - see [discussion below](#))

gamma_n = damping coefficient for collisions in normal direction (1/time units or 1/time-distance units - see discussion below)

gamma_t = damping coefficient for collisions in tangential direction (1/time units or 1/time-distance units - see discussion below)

xmu = static yield criterion (unitless value between 0.0 and 1.0e4)

dampflag = 0 or 1 if tangential damping force is excluded or included

optional keyword = limit_damping, limit damping to prevent attractive interaction

For granular, fstyle_params are set using the same syntax as for the pair_coeff command of [pair_style granular](#)

- wallstyle = *xplane* or *yplane* or *zplane* or *zcylinder*
- args = list of arguments for a particular style

xplane or yplane or zplane args = lo hi
 lo,hi = position of lower and upper plane (distance units), either can be NULL
 zcylinder args = radius
 radius = cylinder radius (distance units)

- zero or more keyword/value pairs may be appended to args
- keyword = *wiggle* or *shear* or *contacts* or *temperature*

wiggle values = dim amplitude period
 dim = x or y or z
 amplitude = size of oscillation (distance units)
 period = time of oscillation (time units)

shear values = dim vshear
 dim = x or y or z
 vshear = magnitude of shear velocity (velocity units)

contacts value = none
 generate contact information for each particle
 temperature value = temperature
 specify temperature of wall

2.259.2 Examples

```
fix 1 all wall/gran hooke 200000.0 NULL 50.0 NULL 0.5 0 xplane -10.0 10.0
fix 1 all wall/gran hooke/history 200000.0 NULL 50.0 NULL 0.5 0 zplane 0.0 NULL
fix 2 all wall/gran hooke 100000.0 20000.0 50.0 30.0 0.5 1 zcylinder 15.0 wiggle z 3.0 2.0
fix 3 all wall/gran granular hooke 1000.0 50.0 tangential linear_nohistory 1.0 0.4 damping velocity region_
→myBox
fix 4 all wall/gran granular jkr 1e5 1500.0 0.3 10.0 tangential mindlin NULL 1.0 0.5 rolling sds 500.0 200.
→0 0.5 twisting marshall region myCone
fix 5 all wall/gran granular dmt 1e5 0.2 0.3 10.0 tangential mindlin NULL 1.0 0.5 rolling sds 500.0 200.0 0.
→5 twisting marshall damping tsuji heat 10 region myCone temperature 1.0
fix 6 all wall/gran hooke 200000.0 NULL 50.0 NULL 0.5 0 xplane -10.0 10.0 contacts
```

2.259.3 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 nature of the wall/particle interactions are determined by the *fstyle* setting. It can be any of the styles defined by the *pair_style gran/** or the more general *pair_style granular* commands. Currently the options are *hooke*, *hooke/history*, or *hertz/history* for the former, and *granular* with all the possible options of the associated *pair_coeff* command for the latter. The equation for the force between the wall and particles touching it is the same as the corresponding equation on the *pair_style gran/** and *pair_style granular* doc pages, in the limit of one of the two particles going to infinite radius and mass (flat wall). Specifically, $\delta = \text{radius} - r$ = overlap of particle with wall, $m_{\text{eff}} = \text{mass of particle}$, and the effective radius of contact = $R_i R_j / (R_i + R_j)$ is set to the radius of the particle.

The parameters *Kn*, *Kt*, *gamma_n*, *gamma_t*, *xmu*, *dampflag*, and the optional keyword *limit_damping* have the same meaning and units as those specified with the *pair_style gran/** commands. This means a NULL can be used for either *Kt* or *gamma_t* as described on that page. If a NULL is used for *Kt*, then a default value is used where $Kt = 2/7 Kn$. If a NULL is used for *gamma_t*, then a default value is used where $gamma_t = 1/2 gamma_n$.

All the model choices for cohesion, tangential friction, rolling friction and twisting friction supported by the *pair_style granular* through its *pair_coeff* command are also supported for walls. These are discussed in greater detail on the doc page for *pair_style granular*.

Note

When *fstyle granular* is specified, the associated *fstyle_params* are taken as those for a wall/particle interaction. For example, for the *hertz/material* normal contact model with $E = 960$ and $\nu = 0.2$, the effective Young's modulus for a wall/particle interaction is computed as $E_{eff} = \frac{960}{2(1-0.2^2)} = 500$. Any pair coefficients defined by *pair_style granular* are not taken into consideration. To model different wall/particle interactions for particles of different material types, the user may define multiple fix wall/gran commands operating on separate groups (e.g. based on particle type) each with a different wall/particle effective Young's modulus.

Note that you can choose a different force styles and/or different values for the wall/particle coefficients than for particle/particle interactions. E.g. if you wish to model the wall as a different material.

Note

As discussed on the page for *pair_style gran/**, versions of LAMMPS before 9Jan09 used a different equation for Hertzian interactions. This means Hertzian wall/particle interactions have also changed. They now include a \sqrt{radius} term which was not present before. Also the previous versions used Kn and Kt from the pairwise interaction and hardwired dampflag to 1, rather than letting them be specified directly. This means you can set the values of the wall/particle coefficients appropriately in the current code to reproduce the results of a previous Hertzian monodisperse calculation. For example, for the common case of a monodisperse system with particles of diameter 1, Kn, Kt, gamma_n, and gamma_s should be set $\sqrt{2.0}$ larger than they were previously.

The effective mass *m_eff* in the formulas listed on the *pair_style granular* page is the mass of the particle for particle/wall interactions (mass of wall is infinite). If the particle is part of a rigid body, its mass is replaced by the mass of the rigid body in those formulas. This is determined by searching for a *fix rigid* command (or its variants).

The *wallstyle* can be planar or cylindrical. The 3 planar options specify a pair of walls in a dimension. Wall positions are given by *lo* and *hi*. Either of the values can be specified as NULL if a single wall is desired. For a *zcylinder* wallstyle, the cylinder's axis is at $x = y = 0.0$, and the radius of the cylinder is specified.

Optionally, the wall can be moving, if the *wiggle* or *shear* keywords are appended. Both keywords cannot be used together.

For the *wiggle* keyword, the wall oscillates sinusoidally, similar to the oscillations of particles which can be specified by the *fix move* command. This is useful in packing simulations of granular particles. The arguments to the *wiggle* keyword specify a dimension for the motion, as well as it's *amplitude* and *period*. Note that if the dimension is in the plane of the wall, this is effectively a shearing motion. If the dimension is perpendicular to the wall, it is more of a shaking motion. A *zcylinder* wall can only be wiggled in the z dimension.

Each timestep, the position of a wiggled wall in the appropriate *dim* is set according to this equation:

position = *coord* + *A* - *A* cos (*omega* * *delta*)

where *coord* is the specified initial position of the wall, *A* is the *amplitude*, *omega* is $2\pi / period$, and *delta* is the time elapsed since the fix was specified. The velocity of the wall is set to the derivative of this expression.

For the *shear* keyword, the wall moves continuously in the specified dimension with velocity *vshear*. The dimension must be tangential to walls with a planar *wallstyle*, e.g. in the *y* or *z* directions for an *xplane* wall. For *zcylinder* walls, a dimension of *z* means the cylinder is moving in the *z*-direction along it's axis. A dimension of *x* or *y* means the cylinder is spinning around the *z*-axis, either in the clockwise direction for *vshear* > 0 or counter-clockwise for *vshear* < 0. In this case, *vshear* is the tangential velocity of the wall at whatever *radius* has been defined.

The *temperature* keyword is used to assign a temperature to the wall. The following value can either be a numeric value or 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 temperature. This

option must be used in conjunction with a heat conduction model defined in [pair_style granular](#), [fix property/atom](#) to store temperature and a heat flow, and [fix heat/flow](#) to integrate heat flow.

Styles with a *gpu*, *intel*, *kk*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed on the [Accelerator packages](#) page. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the GPU, INTEL, KOKKOS, OPENMP, and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the [Build package](#) page for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the [-suffix command-line switch](#) when you invoke LAMMPS, or you can use the [suffix](#) command in your input script.

See the [Accelerator packages](#) page for more instructions on how to use the accelerated styles effectively.

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

This fix writes the shear friction state of atoms interacting with the wall to [binary restart files](#), so that a simulation can continue correctly if granular potentials with shear “history” effects are being used. See the [read_restart](#) command for info on how to re-specify a fix in an input script that reads a restart file, so that the operation of the fix continues in an uninterrupted fashion.

If the contacts option is used, this fix generates a per-atom array with 8 columns as output, containing the contact information for owned particles (nlocal on each processor). All columns in this per-atom array will be zero if no contact has occurred. The values of these columns are listed in the following table:

Index	Value	Units
1	1.0 if particle is in contact with wall, 0.0 otherwise	
2	Force f_x exerted by the wall	force units
3	Force f_y exerted by the wall	force units
4	Force f_z exerted by the wall	force units
5	x -coordinate of contact point on wall	distance units
6	y -coordinate of contact point on wall	distance units
7	z -coordinate of contact point on wall	distance units
8	Radius r of atom	distance units

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](#).

2.259.5 Restrictions

This fix is part of the GRANULAR package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

Any dimension (xyz) that has a granular wall must be non-periodic.

2.259.6 Related commands

fix move, fix wall/gran/region, pair_style gran/ pair_style granular*

2.259.7 Default

none

2.260 fix wall/gran/region command

2.260.1 Syntax

```
fix ID group-ID wall/gran/region fstyle fstyle_params wallstyle regionID keyword values ...
```

- ID, group-ID are documented in [fix](#) command
- wall/region = style name of this fix command
- fstyle = style of force interactions between particles and wall

possible choices: hooke, hooke/history, hertz/history, granular

- fstyle_params = parameters associated with force interaction style

For hooke, hooke/history, and hertz/history, fstyle_params are:

Kn = elastic constant for normal particle repulsion (force/distance units or pressure units - see discussion below)

Kt = elastic constant for tangential contact (force/distance units or pressure units - see discussion below)

gamma_n = damping coefficient for collisions in normal direction (1/time units or 1/time-distance units - see discussion below)

gamma_t = damping coefficient for collisions in tangential direction (1/time units or 1/time-distance units - see discussion below)

xmu = static yield criterion (unitless value between 0.0 and 1.0e4)

dampflag = 0 or 1 if tangential damping force is excluded or included

For granular, fstyle_params are set using the same syntax as for the pair_coeff command of pair_style granular

- wallstyle = region (see [fix wall/gran](#) for options for other kinds of walls)
- region-ID = region whose boundary will act as wall
- keyword = *contacts* or *temperature*

contacts value = none

generate contact information for each particle

temperature value = temperature

specify temperature of wall

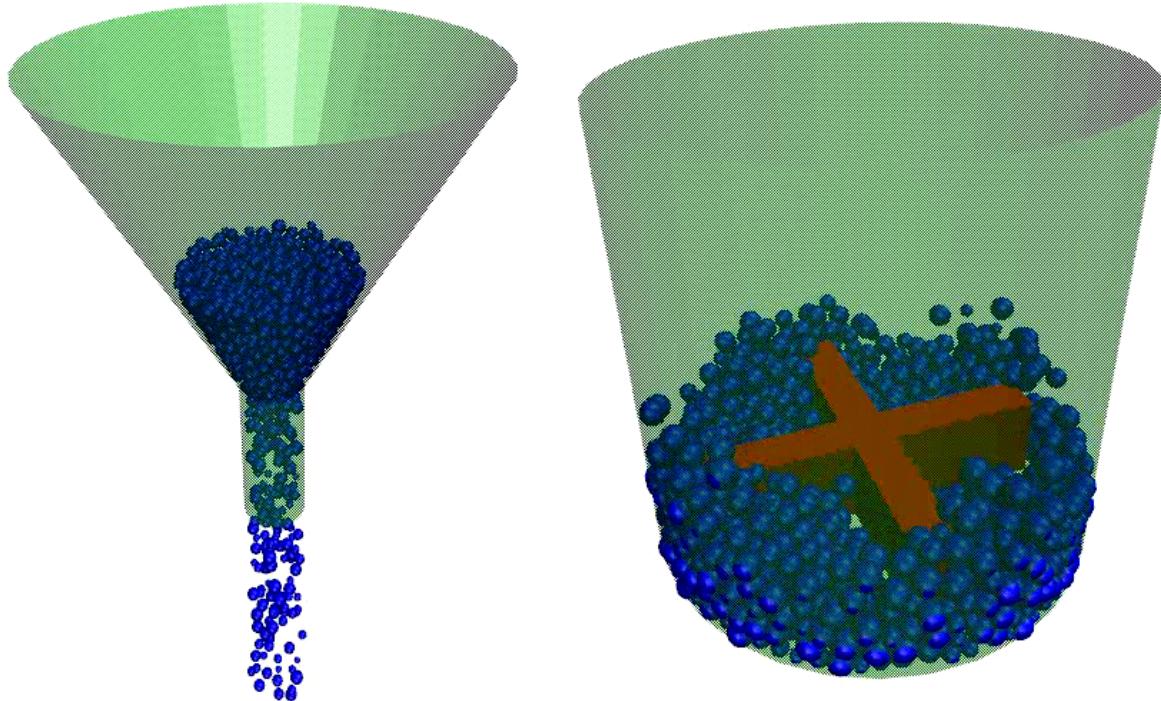
2.260.2 Examples

```
fix wall all wall/gran/region hooke/history 1000.0 200.0 200.0 100.0 0.5 1 region myCone
fix 3 all wall/gran/region granular hooke 1000.0 50.0 tangential linear_nohistory 1.0 0.4 damping velocity
    ↵region myBox
fix 4 all wall/gran/region granular jkr 1e5 1500.0 0.3 10.0 tangential mindlin NULL 1.0 0.5 rolling sds 500.
    ↵0 200.0 0.5 twisting marshall region myCone
fix 5 all wall/gran/region granular dmt 1e5 0.2 0.3 10.0 tangential mindlin NULL 1.0 0.5 rolling sds 500.0
    ↵200.0 0.5 twisting marshall damping tsuji region myCone
fix wall all wall/gran/region hooke/history 1000.0 200.0 200.0 100.0 0.5 1 region myCone contacts
```

2.260.3 Description

Treat the surface of the geometric region defined by the *region-ID* as a bounding frictional wall which interacts with nearby finite-size granular particles when they are close enough to touch the wall. See the [fix wall/region](#) and [fix wall/gran](#) commands for related kinds of walls for non-granular particles and simpler wall geometries, respectively.

Here are snapshots of example models using this command. Corresponding input scripts can be found in examples/granregion. Movies of these simulations are [here](#) on the Movies page of the LAMMPS website.



The distance between a particle and the region boundary is the distance to the nearest point on the region surface. The force the wall exerts on the particle is along the direction between that point and the particle center, which is the direction normal to the surface at that point. Note that if the region surface is comprised of multiple “faces”, then each face can exert a force on the particle if it is close enough. E.g. for [region_style block](#), a particle in the interior, near a corner of the block, could feel wall forces from 1, 2, or 3 faces of the block.

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. The exception to this is if one or more *open* options are specified for the region command, in which case particles interact with both the interior and exterior surfaces of regions.

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 move dynamically via the *region* command keywords (*move*) and *rotate*, or change their shape by use of variables as inputs to the *region* command. If such a region is used with this fix, then the region surface will move in time in the corresponding manner.

Note

As discussed on the *region* command doc page, regions in LAMMPS do not get wrapped across periodic boundaries. It is up to you to ensure that the region location with respect to periodic or non-periodic boundaries is specified appropriately via the *region* and *boundary* commands when using a region as a wall that bounds particle motion.

Note

For primitive regions with sharp corners and/or edges (e.g. a block or cylinder), wall/particle forces are computed accurately for both interior and exterior regions. For *union* and *intersect* regions, additional sharp corners and edges may be present due to the intersection of the surfaces of 2 or more primitive volumes. These corners and edges can be of two types: concave or convex. Concave points/edges are like the corners of a cube as seen by particles in the interior of a cube. Wall/particle forces around these features are computed correctly. Convex points/edges are like the corners of a cube as seen by particles exterior to the cube, i.e. the points jut into the volume where particles are present. LAMMPS does NOT compute the location of these convex points directly, and hence wall/particle forces in the cutoff volume around these points suffer from inaccuracies. The basic problem is that the outward normal of the surface is not continuous at these points. This can cause particles to feel no force (they don't "see" the wall) when in one location, then move a distance epsilon, and suddenly feel a large force because they now "see" the wall. In a worst-case scenario, this can blow particles out of the simulation box. Thus, as a general rule you should not use the fix wall/gran/region command with *union* or *interesect* regions that have convex points or edges resulting from the union/intersection (convex points/edges in the union/intersection due to a single sub-region are still OK).

Note

Similarly, you should not define *union* or *interset* regions for use with this command that share an overlapping common face that is part of the overall outer boundary (interior boundary is OK), even if the face is smooth. E.g. two regions of style block in a *union* region, where the two blocks overlap on one or more of their faces. This is because LAMMPS discards points that are part of multiple sub-regions when calculating wall/particle interactions, to avoid double-counting the interaction. Having two coincident faces could cause the face to become invisible to the particles. The solution is to make the two faces differ by epsilon in their position.

The nature of the wall/particle interactions are determined by the *fstyle* setting. It can be any of the styles defined by the *pair_style gran/** or the more general *pair_style granular* commands. Currently the options are *hooke*, *hooke/history*, or *hertz/history* for the former, and *granular* with all the possible options of the associated *pair_coeff* command for the latter. The equation for the force between the wall and particles touching it is the same as the corresponding equation on the *pair_style gran/** and *pair_style granular* doc pages, but the effective radius is calculated using the radius of the particle and the radius of curvature of the wall at the contact point.

Specifically, $\delta = \text{radius} - r = \text{overlap of particle with wall}$, $m_{\text{eff}} = \text{mass of particle}$, and $R_i R_j / (R_i + R_j)$ is the effective radius, with R_j replaced by the radius of curvature of the wall at the contact point. The radius of curvature can be negative for a concave wall section, e.g. the interior of cylinder. For a flat wall, $\delta = \text{radius} - r = \text{overlap of particle}$

with wall, m_{eff} = mass of particle, and the effective radius of contact is just the radius of the particle.

The parameters Kn , Kt , γ_n , γ_t , xmu , $dampflag$, and the optional keyword *limit damping* have the same meaning and units as those specified with the [pair_style gran/*](#) commands. This means a NULL can be used for either Kt or γ_t as described on that page. If a NULL is used for Kt , then a default value is used where $Kt = 2/7 Kn$. If a NULL is used for γ_t , then a default value is used where $\gamma_t = 1/2 \gamma_n$.

All the model choices for cohesion, tangential friction, rolling friction and twisting friction supported by the [pair_style granular](#) through its *pair_coeff* command are also supported for walls. These are discussed in greater detail on the doc page for [pair_style granular](#).

Note that you can choose a different force styles and/or different values for the 6 wall/particle coefficients than for particle/particle interactions. E.g. if you wish to model the wall as a different material.

The *temperature* keyword is used to assign a temperature to the wall. The following value can either be a numeric value or 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 temperature. This option must be used in conjunction with a heat conduction model defined in [pair_style granular](#), [fix property/atom](#) to store temperature and a heat flow, and [fix heatflow](#) to integrate heat flow.

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

Similar to [fix wall/gran](#) command, this fix writes the shear friction state of atoms interacting with the wall to [binary restart files](#), so that a simulation can continue correctly if granular potentials with shear “history” effects are being used. This fix also includes info about a moving region in the restart file. 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.

Note

Information about region definitions is NOT included in restart files, as discussed on the [read_restart](#) doc page. So you must re-define your region and if it is a moving region, define its motion attributes in a way that is consistent with the simulation that wrote the restart file. In particular, if you want to change the region motion attributes (e.g. its velocity), then you should ensure the position/orientation of the region at the initial restart timestep is the same as it was on the timestep the restart file was written. If this is not possible, you may need to ignore info in the restart file by defining a new fix wall/gran/region command in your restart script, e.g. with a different fix ID. Or if you want to keep the shear history info but discard the region motion information, you can use the same fix ID for fix wall/gran/region, but assign it a region with a different region ID.

If the contacts option is used, this fix generates a per-atom array with 8 columns as output, containing the contact information for owned particles (nlocal on each processor). All columns in this per-atom array will be zero if no contact has occurred. The values of these columns are listed in the following table:

Index	Value	Units
1	1.0 if particle is in contact with wall, 0.0 otherwise	
2	Force f_x exerted by the wall	force units
3	Force f_y exerted by the wall	force units
4	Force f_z exerted by the wall	force units
5	x -coordinate of contact point on wall	distance units
6	y -coordinate of contact point on wall	distance units
7	z -coordinate of contact point on wall	distance units
8	Radius r of atom	distance units

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*.

2.260.5 Restrictions

This fix is part of the GRANULAR package. It is only enabled if LAMMPS was built with that package. See the *Build package* page for more info.

2.260.6 Related commands

fix_move, fix wall/gran, fix wall/region, pair_style granular, region

2.260.7 Default

none

2.261 fix wall/piston command

2.261.1 Syntax

```
fix ID group-ID wall/piston face ... keyword value ...
```

- ID, group-ID are documented in *fix* command
 - wall/piston = style name of this fix command
 - face = *zlo*
 - zero or more keyword/value pairs may be appended
 - keyword = *pos* or *vel* or *ramp* or *temp* or *units*
- pos args = *z*
z = *z* coordinate at which the piston begins (distance units)
- vel args = *vz*
vz = final velocity of the piston (velocity units)
- ramp* = use a linear velocity ramp from 0 to *vz*
- temp args = target damp seed extent
target = target velocity for region immediately ahead of the piston
damp = damping parameter (time units)
seed = random number seed for langevin kicks
extent = extent of thermostatted region (distance 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

2.261.2 Examples

```
fix xwalls all wall/piston zlo
fix walls all wall/piston zlo pos 1.0 vel 10.0 units box
fix top all wall/piston zlo vel 10.0 ramp
```

2.261.3 Description

Bound the simulation with a moving wall which reflect particles in the specified group and drive the system with an effective infinite-mass piston capable of driving shock waves.

A momentum mirror technique is used, which means that if an atom (or the wall) moves such that an atom is 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 velocity relative to the moving wall is flipped in z. For instance, a stationary particle hit with a piston wall with velocity vz, will end the timestep with a velocity of 2*vz.

Currently the *face* keyword can only be *zlo*. This creates a piston moving in the positive z direction. Particles with z coordinate less than the wall position are reflected to a z coordinate greater than the wall position. If the piston velocity is vpz and the particle velocity before reflection is vzi, the particle velocity after reflection is -vzi + 2*vpz.

The initial position of the wall can be specified by the *pos* keyword.

The final velocity of the wall can be specified by the *vel* keyword

The *ramp* keyword will cause the wall/piston to adjust the velocity linearly from zero velocity to *vel* over the course of the run. If the *ramp* keyword is omitted then the wall/piston moves at a constant velocity defined by *vel*.

The *temp* keyword will cause the region immediately in front of the wall/piston to be thermostatted with a Langevin thermostat. This region moves with the piston. The damping and kicking are measured in the reference frame of the piston. So, a temperature of zero would mean all particles were moving at exactly the speed of the wall/piston.

The *units* keyword determines the meaning of the distance units used to define a wall position, but only when a numeric constant is 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 spacings.

2.261.4 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](#).

2.261.5 Restrictions

This fix style is part of the SHOCK package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

The face that has the wall/piston must be boundary type ‘s’ (shrink-wrapped). The opposing face can be any boundary type other than periodic.

A wall/piston should not be used with rigid bodies such as those defined by a “fix rigid” command. This is because the wall/piston displaces atoms directly rather than exerting a force on them.

2.261.6 Related commands

fix wall/reflect command, *fix append/atoms* command

2.261.7 Default

The keyword defaults are pos = 0, vel = 0, units = lattice.

2.262 fix wall/reflect command

Accelerator Variants: *wall/reflect/kk*

2.262.1 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*

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

- 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

2.262.2 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
```

2.262.3 Description

Bound the simulation with one or more walls which reflect particles in the specified group when they attempt to move through 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(dt²) 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(0.0,5.0,0.3,0)
fix 1 all wall/reflect xlo v_wiggle

variable wiggle equal cwiggle(0.0,5.0,0.3,0)
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:

$$position = c0 + A (1 - \cos(\omega * delta))$$

Styles with a *gpu*, *intel*, *kk*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed on the [Accelerator packages](#) page. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the GPU, INTEL, KOKKOS, OPENMP, and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the [Build package](#) page for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the `-suffix command-line switch` when you invoke LAMMPS, or you can use the `suffix` command in your input script.

See the [Accelerator packages](#) page for more instructions on how to use the accelerated styles effectively.

2.262.4 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](#).

2.262.5 Restrictions

Any dimension (xyz) that has a reflecting wall must be non-periodic.

A reflecting wall should not be used with rigid bodies such as those defined by a “fix rigid” command. This is because the wall/reflect displaces atoms directly rather than exerts a force on them. For rigid bodies, use a soft wall instead, such as [fix wall/lj93](#). LAMMPS will flag the use of a rigid fix with `fix wall/reflect` with a warning, but will not generate an error.

2.262.6 Related commands

[fix wall/lj93](#), [fix oneway](#)

2.262.7 Default

The default for the units keyword is lattice.

(**Bond**) Bond and Leimkuhler, SIAM J Sci Comput, 30, p 134 (2007).

2.263 fix wall/reflect/stochastic command

2.263.1 Syntax

```
fix ID group-ID wall/reflect/stochastic rstyle seed face args ... keyword value ...
```

- ID, group-ID are documented in [fix](#) command
- wall/reflect/stochastic = style name of this fix command
- rstyle = diffusive or maxwell or ccl
- seed = random seed for stochasticity (positive integer)
- one or more face/args pairs may be appended
- face = *xlo* or *xhi* or *ylo* or *yhi* or *zlo* or *zhi*

args = pos temp velx vely velz accomx accomy accomz
pos = EDGE or constant
EDGE = current lo or hi edge of simulation box
constant = number like 0.0 or 30.0 (distance units)
temp = wall temperature (temperature units)
velx,vely,velz = wall velocity in x,y,z directions (velocity units)
accomx,accomm,accomz = accommodation coeffs in x,y,z directions (unitless)
not specified for rstyle = diffusive
single accom coeff specified for rstyle maxwell
all 3 coeffs specified for rstyle cll

- 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

2.263.2 Examples

```
fix zwalls all wall/reflect/stochastic diffusive 23424 zlo EDGE 300 0.1 0.1 0 zhi EDGE 200 0.1 0.1 0
fix ywalls all wall/reflect/stochastic maxwell 345533 ylo 5.0 300 0.1 0.0 0.0 0.8 yhi 10.0 300 0.1 0.0 0.0 0.8
fix xwalls all wall/reflect/stochastic cercignanilampis 2308 xlo 0.0 300 0.0 0.1 0.9 0.8 0.7 xhi EDGE 300 0.
→ 0 0.1 0 0.9 0.8 0.7 units box
```

2.263.3 Description

Bound the simulation with one or more walls which reflect particles in the specified group when they attempt to move through them.

Reflection means that if an atom moves outside the wall on a timestep (e.g. due to the [fix nve](#) command), then it is put back inside the wall with a changed velocity.

This fix models treats the wall as a moving solid boundary with a finite temperature, which can exchange energy with particles that collide with it. This is different than the simpler [fix wall/reflect](#) command which models mirror reflection. For this fix, the post collision velocity of each particle is treated stochastically. The randomness can come from many sources: thermal motion of the wall atoms, surface roughness, etc. Three stochastic reflection models are currently implemented.

For rstyle *diffusive*, particles are reflected diffusively. Their velocity distribution corresponds to an equilibrium distribution of particles at the wall temperature. No accommodation coefficients are specified.

For rstyle *maxwell*, particle reflection is Maxwellian which means partially diffusive and partially specular ([Maxwell](#)). A single accommodation coeff is specified which must be between 0.0 and 1.0 inclusive. It determines the fraction of the collision which is diffusive versus specular. An accommodation coefficient of 1.0 is fully diffusive; a coefficient of 0.0 is fully specular.

For rstyle *cll*, particle collisions are computed by the Cercignani/Lampis model. See [CL](#) and [To](#) for details. Three accommodations coefficient are specified. Each must be between 0.0 and 1.0 inclusive. Two are velocity accommodation coefficients; one is a normal kinetic energy accommodation. The normal coeff is the one corresponding to the normal of the wall itself. For example if the wall is *ylo* or *yhi*, *accomx* and *accomz* are the tangential velocity accommodation coefficients, and *accomy* is the normal kinetic energy accommodation coefficient.

The optional *units* keyword determines the distance units used to define a wall position. A *box* value selects standard distance units as defined by the [units](#) command, e.g. Angstroms for units = real or metal. A *lattice* value means the distance units are in lattice spacings. The [lattice](#) command must have been previously used to define the lattice spacings.

2.263.4 Restrictions

This fix has the same limitations as the [fix wall/reflect](#) command. Any dimension (xyz) that has a wall must be non-periodic. It should not be used with rigid bodies such as those defined by the [fix rigid](#) command. The wall velocity must lie on the same plane as the wall itself.

This fix is part of the EXTRA-FIX package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) page for more info.

2.263.5 Related commands

[fix wall/reflect](#)

2.263.6 Default

The default for the units keyword is lattice.

(Maxwell) J.C. Maxwell, Philos. Tans. Royal Soc. London, 157: 49-88 (1867).

(Cercignani) C. Cercignani and M. Lampis. Trans. Theory Stat. Phys. 1, 2, 101 (1971).

(To) Q.D. To, V.H. Vu, G. Lauriat, and C. Leonard. J. Math. Phys. 56, 103101 (2015).

2.264 fix wall/region command

2.264.1 Syntax

```
fix ID group-ID wall/region region-ID style args ... 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 *lj1043* or *colloid* or *harmonic* or *morse*
- args for styles *lj93* or *lj126* or *lj1043* or *colloid* or *harmonic* =

epsilon = strength factor for wall-particle interaction (energy or energy/distance² units)

sigma = size factor for wall-particle interaction (distance units)
- args for style *morse* =

D_0 = depth of the potential (energy units)

alpha = width parameter (1/distance units)

r_0 = distance of the potential minimum from wall position (distance units)
- cutoff = distance from wall at which wall-particle interaction is cut off (distance units)

2.264.2 Examples

```
fix wall all wall/region mySphere lj93 1.0 1.0 2.5
fix wall all wall/region mySphere harmonic 1.0 0.0 2.5
fix wall all wall/region box_top morse 1.0 1.0 1.5 3.0
```

2.264.3 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. Note that if the region surface is comprised of multiple “faces”, then each face can exert a force on the particle if it is close enough. E.g. for *region_style block*, a particle in the interior, near a corner of the block, could feel wall forces from 1, 2, or 3 faces of the block.

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. The surface of the region only exerts forces on particles “inside” the region; if a particle is “outside” the region it will generate an error, because it has moved through the wall.

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.

Note

As discussed on the [region](#) command doc page, regions in LAMMPS do not get wrapped across periodic boundaries. It is up to you to ensure 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.

Note

For primitive regions with sharp corners and/or edges (e.g. a block or cylinder), wall/particle forces are computed accurately for both interior and exterior regions. For *union* and *intersect* regions, additional sharp corners and edges may be present due to the intersection of the surfaces of 2 or more primitive volumes. These corners and edges can be of two types: concave or convex. Concave points/edges are like the corners of a cube as seen by particles in the interior of a cube. Wall/particle forces around these features are computed correctly. Convex points/edges are like the corners of a cube as seen by particles exterior to the cube, i.e. the points jut into the volume where particles are present. LAMMPS does NOT compute the location of these convex points directly, and hence wall/particle forces in the cutoff volume around these points suffer from inaccuracies. The basic problem is that the outward normal of the surface is not continuous at these points. This can cause particles to feel no force (they don’t “see” the wall) when in one location, then move a distance epsilon, and suddenly feel a large force because they now “see” the wall. In a worst-case scenario, this can blow particles out of the simulation box. Thus, as a general rule you should not use the fix wall/gran/region command with *union* or *intersect* regions that have convex points or edges resulting from the union/intersection (convex points/edges in the union/intersection due to a single sub-region are still OK).

Note

Similarly, you should not define *union* or *intersect* regions for use with this command that share an overlapping common face that is part of the overall outer boundary (interior boundary is OK), even if the face is smooth. E.g. two regions of style block in a *union* region, where the two blocks overlap on one or more of their faces. This is because LAMMPS discards points that are part of multiple sub-regions when calculating wall/particle interactions, to avoid double-counting the interaction. Having two coincident faces could cause the face to become invisible to the particles. The solution is to make the two faces differ by epsilon in their position.

The energy of wall-particle interactions depends on the specified style.

For style *lj93*, the energy E is given by the 9/3 potential:

$$E = \epsilon \left[\frac{2}{15} \left(\frac{\sigma}{r} \right)^9 - \left(\frac{\sigma}{r} \right)^3 \right] \quad r < r_c$$

For style *lj126*, the energy E is given by the 12/6 potential:

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

For style *wall/lj1043*, the energy E is given by the 10/4/3 potential:

$$E = 2\pi\epsilon \left[\frac{2}{5} \left(\frac{\sigma}{r} \right)^{10} - \left(\frac{\sigma}{r} \right)^4 - \frac{\sqrt{2}\sigma^3}{3(r + (0.61/\sqrt{2})\sigma)^3} \right] \quad r < r_c$$

For style *colloid*, the energy E is given by an integrated form of the [pair_style colloid](#) potential:

$$E = \epsilon \left[\frac{\sigma^6}{7560} \left(\frac{6R - D}{D^7} + \frac{D + 8R}{(D + 2R)^7} \right) - \frac{1}{6} \left(\frac{2R(D + R) + D(D + 2R)[\ln D - \ln(D + 2R)]}{D(D + 2R)} \right) \right] \quad r < r_c$$

For style *wall/harmonic*, the energy E is given by a harmonic spring potential (the distance parameter is ignored):

$$E = \epsilon (r - r_c)^2 \quad r < r_c$$

For style *wall/morse*, the energy E is given by the Morse potential:

$$E = D_0 [e^{-2\alpha(r-r_0)} - 2e^{-\alpha(r-r_0)}] \quad r < r_c$$

Unlike other styles, this requires three parameters (D_0 , α , and r_0 in this order) instead of two like for the other wall styles.

In all cases, r is the distance from the particle to the region surface, and R_c is the *cutoff* distance at which the particle and surface no longer interact. The cutoff is always the last argument. The energy of the wall potential is shifted so that the wall-particle interaction energy is 0.0 at the cutoff distance.

For a full description of these wall styles, see [fix_style wall](#)

2.264.4 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 region wall to the global potential energy of the system as part of [thermodynamic output](#). The default setting for this fix is [fix_modify energy no](#).

The [fix_modify virial](#) option is supported by this fix to add the contribution due to the interaction between atoms and the region wall to both the global pressure and per-atom stress of the system via the [compute pressure](#) and [compute stress/atom](#) commands. The former can be accessed by [thermodynamic output](#). The default setting for this fix is [fix_modify virial no](#).

The [fix_modify respa](#) option is supported by this fix. This allows to set at which level of the [r-RESPA](#) integrator the fix is adding its forces. Default is the outermost level.

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.

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.

2.264.5 Restrictions

none

2.264.6 Related commands

[fix wall/lj93](#), [fix wall/lj126](#), [fix wall/lj1043](#), [fix wall/colloid](#), [fix wall/harmonic](#), [fix wall/gran](#)

2.264.7 Default

none

2.265 fix wall/srd command

2.265.1 Syntax

`fix ID group-ID wall/srd face arg ... keyword value ...`

- ID, group-ID are documented in [fix](#) command
- wall/srd = style name of this fix command
- one or more face/arg pairs may be appended
- face = *xlo* or *xhi* or *ylo* or *yhi* or *zlo* or *zhi*

xlo,ylo,zlo arg = EDGE or constant or variable
EDGE = current lo edge of simulation box
constant = number like 0.0 or -30.0 (distance units)
variable = [equal-style variable](#) like v_x or v_wiggle
xhi,yhi,zhi arg = EDGE or constant or variable
EDGE = current hi edge of simulation box
constant = number like 50.0 or 100.3 (distance units)
variable = [equal-style variable](#) like v_x or v_wiggle

- zero or more keyword/value pairs may be appended
 - keyword = *units*
- units value* = lattice or box
lattice = the wall position is defined in lattice units
box = the wall position is defined in simulation box units

2.265.2 Examples

```
fix xwalls all wall/srd xlo EDGE xhi EDGE
fix walls all wall/srd xlo 0.0 ylo 10.0 units box
fix top all wall/srd zhi v_pressdown
```

2.265.3 Description

Bound the simulation with one or more walls which interact with stochastic reaction dynamics (SRD) particles as slip (smooth) or no-slip (rough) flat surfaces. The wall interaction is actually invoked via the [fix srd](#) command, only on the group of SRD particles it defines, so the group setting for the fix wall/srd command is ignored.

A particle/wall collision occurs if an SRD particle moves outside the wall on a timestep. This alters the position and velocity of the SRD particle and imparts a force to the wall.

The *collision* and *Tsrd* settings specified via the [fix srd](#) command affect the SRD/wall collisions. A *slip* setting for the *collision* keyword means that the tangential component of the SRD particle momentum is preserved. Thus only a normal force is imparted to the wall. The normal component of the new SRD velocity is sampled from a Gaussian distribution at temperature *Tsrd*.

For a *noslip* setting of the *collision* keyword, both the normal and tangential components of the new SRD velocity are sampled from a Gaussian distribution at temperature *Tsrd*. Additionally, a new tangential direction for the SRD velocity is chosen randomly. This collision style imparts both a normal and tangential force to the wall.

Up to 6 walls or faces can be specified in a single command: *xlo*, *xhi*, *ylo*, *yhi*, *zlo*, *zhi*. A *lo* face reflects particles that move to a coordinate less than the wall position, back in the *hi* direction. A *hi* face reflects particles that move to a coordinate higher than the wall position, back in the *lo* direction.

The position of each wall can be specified in one of 3 ways: as the EDGE of the simulation box, as a constant value, or as a variable. If EDGE is used, then the corresponding boundary of the current simulation box is used. If a numeric constant is specified then the wall is placed at that position in the appropriate dimension (x, y, or z). In both the EDGE and constant cases, the wall will never move. If the wall position is a variable, it should be specified as *v_name*, where *name* is an [equal-style variable](#) name. In this case the variable is evaluated each timestep and the result becomes the current position of the reflecting wall. Equal-style variables can specify formulas with various mathematical functions, and include [thermo_style](#) command keywords for the simulation box parameters and timestep and elapsed time. Thus it is easy to specify a time-dependent wall position.

Note

Because the trajectory of the SRD particle is tracked as it collides with the wall, you must ensure that $r = \text{distance}$ of the particle from the wall, is always > 0 for SRD particles, or LAMMPS will generate an error. This means you cannot start your simulation with SRD particles at the wall position *coord* ($r = 0$) or with particles on the wrong side of the wall ($r < 0$).

Note

If you have 2 or more walls that come together at an edge or corner (e.g. walls in the x and y dimensions), then be sure to set the *overlap* keyword to *yes* in the [fix srd](#) command, since the walls effectively overlap when SRD particles collide with them. LAMMPS will issue a warning if you do not do this.

Note

The walls of this fix only interact with SRD particles, as defined by the [fix srd](#) command. If you are simulating a mixture containing other kinds of particles, then you should typically use [another wall command](#) to act on the other particles. Since SRD particles will be colliding both with the walls and the other particles, it is important to ensure that the other particle's finite extent does not overlap an SRD wall. If you do not do this, you may generate errors when SRD particles end up “inside” another particle or a wall at the beginning of a collision step.

The *units* keyword determines the meaning of the distance units used to define a wall position, but only when a numeric constant is used. It is not relevant when EDGE or a variable is used to specify a face position.

A *box* value selects standard distance units as defined by the [units](#) command, e.g. Angstroms for units = real or metal. A *lattice* value means the distance units are in lattice spacings. The [lattice](#) command must have been previously used to define the lattice spacings.

Here are examples of variable definitions that move the wall position in a time-dependent fashion using equal-style *variables*.

```
variable ramp equal ramp(0,10)
fix 1 all wall/srd xlo v_ramp

variable linear equal vdisplace(0,20)
fix 1 all wall/srd xlo v_linear

variable wiggle equal swiggle(0.0,5.0,3.0)
fix 1 all wall/srd xlo v_wiggle

variable wiggle equal cwiggle(0.0,5.0,3.0)
fix 1 all wall/srd 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*:

$$\text{position} = \text{c0} + \text{A} \sin(\text{omega} * \text{delta})$$

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:

$$\text{position} = \text{c0} + \text{A} (1 - \cos(\text{omega} * \text{delta}))$$

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

No information about this fix is written to *binary restart files*. None of the *fix_modify* options are relevant to this fix.

This fix computes a global array of values which can be accessed by various *output commands*. The number of rows in the array is equal to the number of walls defined by the fix. The number of columns is 3, for the x,y,z components of force on each wall.

Note that an outward normal force on a wall will be a negative value for *lo* walls and a positive value for *hi* walls. The array values calculated by this fix are “extensive”.

No parameter of this fix can be used with the *start/stop* keywords of the *run* command. This fix is not invoked during *energy minimization*.

2.265.5 Restrictions

Any dimension (xyz) that has an SRD wall must be non-periodic.

2.265.6 Related commands

fix srd

2.265.7 Default

none

2.266 fix widom command

2.266.1 Syntax

```
fix ID group-ID widom N M type seed T keyword values ...
```

- ID, group-ID are documented in *fix* command
- widom = style name of this fix command
- N = invoke this fix every N steps
- M = number of Widom insertions to attempt every N steps
- type = atom type (1-Ntypes or type label) for inserted atoms (must be 0 if mol keyword used)
- seed = random # seed (positive integer)
- T = temperature of the system (temperature units)
- zero or more keyword/value pairs may be appended to args

keyword = mol, region, full_energy, charge, intra_energy

mol value = template-ID

template-ID = ID of molecule template specified in a separate molecule command
region value = region-ID

region-ID = ID of region where Widom insertions are allowed

full_energy = compute the entire system energy when performing Widom insertions

charge value = charge of inserted atoms (charge units)
intra_energy value = intramolecular energy (energy units)

2.266.2 Examples

```
fix 2 gas widom 1 50000 1 19494 2.0
fix 3 water widom 1000 100 0 29494 300.0 mol h2omol full_energy

labelmap atom 1 Li
fix 2 ion widom 1 50000 Li 19494 2.0
```

2.266.3 Description

This fix performs Widom insertions of atoms or molecules at the given temperature as discussed in ([Frenkel](#)). Specific uses include computation of Henry constants of small molecules in microporous materials or amorphous systems.

Every N timesteps the fix attempts M number of Widom insertions of atoms or molecules.

If the *mol* keyword is used, only molecule insertions are performed. Conversely, if the *mol* keyword is not used, only atom insertions are performed.

This command may optionally use the *region* keyword to define an insertion volume. The specified region must have been previously defined with a [region](#) command. It must be defined with side = *in*. Insertion attempts occur only within the specified region. For non-rectangular regions, random trial points are generated within the rectangular bounding box until a point is found that lies inside the region. If no valid point is generated after 1000 trials, no insertion is performed. If an attempted insertion places the atom or molecule center-of-mass outside the specified region, a new attempted insertion is generated. This process is repeated until the atom or molecule center-of-mass is inside the specified region.

Note that neighbor lists are re-built every timestep that this fix is invoked, so you should not set N to be too small. See the [neighbor](#) command for details.

When an atom or molecule is to be inserted, its coordinates are chosen at a random position within the current simulation cell or region. Relative coordinates for atoms in a molecule are taken from the template molecule provided by the user. The center of mass of the molecule is placed at the insertion point. The orientation of the molecule is chosen at random by rotating about this point.

Individual atoms are inserted, unless the *mol* keyword is used. It specifies a *template-ID* previously defined using the [molecule](#) command, which reads a file that defines the molecule. The coordinates, atom types, charges, etc., as well as any bonding and special neighbor information for the molecule can be specified in the molecule file. See the [molecule](#) command for details. The only settings required to be in this file are the coordinates and types of atoms in the molecule.

Note that fix widom does not use configurational bias MC or any other kind of sampling of intramolecular degrees of freedom. Inserted molecules can have different orientations, but they will all have the same intramolecular configuration, which was specified in the molecule command input.

For atoms, inserted particles have the specified atom type. For molecules, they use the same atom types as in the template molecule supplied by the user.

The excess chemical potential μ_{ex} is defined as:

$$\mu_{ex} = -kT \ln(<\exp(-(U_{N+1} - U_N)/k_B T)>)$$

where k_B is the Boltzmann constant, T is the user-specified temperature, U_N and U_{N+1} is the potential energy of the system with N and $N + 1$ particles.

The *full_energy* option means that the fix calculates the total potential energy of the entire simulated system, instead of just the energy of the part that is changed. By default, this option is off, in which case only partial energies are computed to determine the energy difference due to the proposed change.

The *full_energy* option is needed for systems with complicated potential energy calculations, including the following:

- long-range electrostatics (kspace)
- many-body pair styles
- hybrid pair styles
- eam pair styles
- tail corrections
- need to include potential energy contributions from other fixes

In these cases, LAMMPS will automatically apply the *full_energy* keyword and issue a warning message.

When the *mol* keyword is used, the *full_energy* option also includes the intramolecular energy of inserted and deleted molecules, whereas this energy is not included when *full_energy* is not used. If this is not desired, the *intra_energy* keyword can be used to define an amount of energy that is subtracted from the final energy when a molecule is inserted, and subtracted from the initial energy when a molecule is deleted. For molecules that have a non-zero intramolecular energy, this will ensure roughly the same behavior whether or not the *full_energy* option is used.

Some fixes have an associated potential energy. Examples of such fixes include: *efield*, *gravity*, *addforce*, *restrain*, and *wall fixes*. For that energy to be included in the total potential energy of the system (the quantity used when performing Widom insertions), you MUST enable the *fix_modify energy* option for that fix. The doc pages for individual *fix* commands specify if this should be done.

Use the *charge* option to insert atoms with a user-specified point charge. Note that doing so will cause the system to become non-neutral. LAMMPS issues a warning when using long-range electrostatics (kspace) with non-neutral systems. See the *compute group/group* documentation for more details about simulating non-neutral systems with kspace on.

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

This fix writes the state of the fix to *binary restart files*. This includes information about the random number generator seed, the next timestep for Widom insertions etc. See the *read_restart* command for info on how to re-specify a fix in an input script that reads a restart file, so that the operation of the fix continues in an uninterrupted fashion.

Note

For this to work correctly, the timestep must **not** be changed after reading the restart with *reset_timestep*. The fix will try to detect it and stop with an error.

None of the *fix_modify* options are relevant to this fix.

This fix computes a global vector of length 3, which can be accessed by various *output commands*. The vector values are the following global cumulative quantities:

1. average excess chemical potential on each timestep
2. average difference in potential energy on each timestep
3. volume of the insertion region

The vector values calculated by this fix are “intensive”.

No parameter of this fix can be used with the *start/stop* keywords of the [run](#) command. This fix is not invoked during *energy minimization*.

2.266.5 Restrictions

This fix is part of the MC package. It is only enabled if LAMMPS was built with that package. See the [Build package](#) doc page for more info.

Do not set “neigh_modify once yes” or else this fix will never be called. Renighboring is **required**.

This fix style requires an [atom style](#) with per atom type masses.

Can be run in parallel, but some aspects of the insertion procedure will not scale well in parallel. Only usable for 3D simulations.

2.266.6 Related commands

fix gcmc *fix atom/swap*, *neighbor*, *fix deposit*, *fix evaporate*,

2.266.7 Default

The option defaults are mol = no, intra_energy = 0.0 and full_energy = no, except for the situations where full_energy is required, as listed above.

(**Frenkel**) Frenkel and Smit, Understanding Molecular Simulation, Academic Press, London, 2002.