CHAPTER

THREE

COMPUTES

3.1 compute ackland/atom command

3.1.1 Syntax

```
compute ID group-ID ackland/atom keyword/value
```

- ID, group-ID are documented in *compute* command
- ackland/atom = style name of this compute command
- zero or more keyword/value pairs may be appended
- keyword = *legacy*

  legacy args = yes or no = use (yes) or do not use (no) legacy Ackland algorithm implementation

3.1.2 Examples

```
compute 1 all ackland/atom
compute 1 all ackland/atom legacy yes
```

3.1.3 Description

Defines a computation that calculates the local lattice structure according to the formulation given in *(Ackland)*. Historically, LAMMPS had two, slightly different implementations of the algorithm from the paper. With the *legacy* keyword, it is possible to switch between the pre-2015 (*legacy yes*) and post-2015 implementation (*legacy no*). The post-2015 variant is the default.

In contrast to the *centro-symmetry parameter* this method is stable against temperature boost, because it is based not on the distance between particles but the angles. Therefore statistical fluctuations are averaged out a little more. A comparison with the Common Neighbor Analysis metric is made in the paper.

The result is a number which is mapped to the following different lattice structures:

- 0 = UNKNOWN
- 1 = BCC
- 2 = FCC
- 3 = HCP

- 4 = ICO

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

3.1.4 Output info

This compute calculates a per-atom vector, which can be accessed by any command that uses per-atom values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

3.1.5 Restrictions

This compute is part of the EXTRA-COMPUTE package. It is only enabled if LAMMPS was built with that package. See the *Build package* page for more info.

The per-atom vector values will be unitless since they are the integers defined above.

3.1.6 Related commands

compute centro/atom

3.1.7 Default

The keyword *legacy* defaults to *no*.

(Ackland) Ackland, Jones, Phys Rev B, 73, 054104 (2006).

3.2 compute adf command

3.2.1 Syntax

```
compute ID group-ID adf Nbin itype1 jtype1 ktype1 Rjinner1 Rjouter1 Rkinner1 Rkouter1 ...
```

- ID, group-ID are documented in *compute* command
- adf = style name of this compute command
- Nbin = number of ADF bins
- itypeN = central atom type for Nth ADF histogram (see asterisk form below)
- jtypeN = J atom type for Nth ADF histogram (see asterisk form below)
- ktypeN = K atom type for Nth ADF histogram (see asterisk form below)
- RjinnerN = inner radius of J atom shell for Nth ADF histogram (distance units)
- RjouterN = outer radius of J atom shell for Nth ADF histogram (distance units)
- RkinnerN = inner radius of K atom shell for Nth ADF histogram (distance units)
- RkouterN = outer radius of K atom shell for Nth ADF histogram (distance units)

- zero or one keyword/value pairs may be appended
- keyword = *ordinate*

  ordinate value = degree or radian or cosine
    Choose the ordinate parameter for the histogram

3.2.2 Examples

```
compute 1 fluid adf 32 1 1 1 0.0 1.2 0.0 1.2 &
                    1 1 2 0.0 1.2 0.0 1.5 &
                    1 2 2 0.0 1.5 0.0 1.5 &
                    2 1 1 0.0 1.2 0.0 1.2 &
                    2 1 2 0.0 1.5 2.0 3.5 &
                    2 2 2 2.0 3.5 2.0 3.5
compute 1 fluid adf 32 1*2 1*2 1*2 0.5 3.5
compute 1 fluid adf 32
```

3.2.3 Description

Define a computation that calculates one or more angular distribution functions (ADF) for a group of particles. Each ADF is calculated in histogram form by measuring the angle formed by a central atom and two neighbor atoms and binning these angles into *Nbin* bins. Only neighbors for which *Rinner* < *R* < *Router* are counted, where *Rinner* and *Router* are specified separately for the first and second neighbor atom in each requested ADF.

Note

If you have a bonded system, then the settings of *special_bonds* command can remove pairwise interactions between atoms in the same bond, angle, or dihedral. This is the default setting for the *special_bonds* command, and means those pairwise interactions do not appear in the neighbor list. Because this fix uses a neighbor list, it also means those pairs will not be included in the ADF. This does not apply when using long-range coulomb interactions (*coul/long*, *coul/msm*, *coul/wolf* or similar. One way to get around this would be to set special_bond scaling factors to very tiny numbers that are not exactly zero (e.g. 1.0e-50). Another workaround is to write a dump file, and use the *rerun* command to compute the ADF for snapshots in the dump file. The rerun script can use a *special_bonds* command that includes all pairs in the neighbor list.

Note

If you request any outer cutoff *Router* > force cutoff, or if no pair style is defined, e.g. the *rerun* command is being used to post-process a dump file of snapshots you must ensure ghost atom information out to the largest value of *Router* + *skin* is communicated, via the *comm_modify cutoff* command, else the ADF computation cannot be performed, and LAMMPS will give an error message. The *skin* value is what is specified with the *neighbor* command.

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

If *itypeN*, *jtypeN*, and *ktypeN* are single values, as in the first example above, this means that the ADF is computed where atoms of type *itypeN* are the central atom, and neighbor atoms of type *jtypeN* and *ktypeN* are forming the angle. If any of *itypeN*, *jtypeN*, or *ktypeN* represent a range of values via the wild-card asterisk, as in the second example above, this means that the ADF is computed where atoms of any of the range of types represented by *itypeN* are the central atom, and the angle is formed by two neighbors, one neighbor in the range of types represented by *jtypeN* and another neighbor in the range of types represented by *ktypeN*.

If no *itypeN*, *jtypeN*, *ktypeN* settings are specified, then LAMMPS will generate a single ADF for all atoms in the group. The inner cutoff is set to zero and the outer cutoff is set to the force cutoff. If no pair_style is specified, there is no force cutoff and LAMMPS will give an error message. Note that in most cases, generating an ADF for all atoms is not a good thing. Such an ADF is both uninformative and extremely expensive to compute. For example, with liquid water with a 10 A force cutoff, there are 80,000 angles per atom. In addition, most of the interesting angular structure occurs for neighbors that are the closest to the central atom, involving just a few dozen angles.

Angles for each ADF are generated by double-looping over the list of neighbors of each central atom I, just as they would be in the force calculation for a three-body potential such as *Stillinger-Weber*. The angle formed by central atom I and neighbor atoms J and K is included in an ADF if the following criteria are met:

- atoms I,J,K are all in the specified compute group
- the distance between atoms I,J is between Rjinner and Rjouter
- the distance between atoms I,K is between Rkinner and Rkouter
- the type of the I atom matches itypeN (one or a range of types)
- atoms I,J,K are distinct
- the type of the J atom matches jtypeN (one or a range of types)
- the type of the K atom matches ktypeN (one or a range of types)

Each unique angle satisfying the above criteria is counted only once, regardless of whether either or both of the neighbor atoms making up the angle appear in both the J and K lists. It is OK if a particular angle is included in more than one individual histogram, due to the way the *itypeN*, *jtypeN*, *ktypeN* arguments are specified.

The first ADF value for a bin is calculated from the histogram count by dividing by the total number of triples satisfying the criteria, so that the integral of the ADF w.r.t. angle is 1, i.e. the ADF is a probability density function.

The second ADF value is reported as a cumulative sum of all bins up to the current bins, averaged over atoms of type *itypeN*. It represents the number of angles per central atom with angle less than or equal to the angle of the current bin, analogous to the coordination number radial distribution function.

The *ordinate* optional keyword determines whether the bins are of uniform angular size from zero to 180 (*degree*), zero to Pi (*radian*), or the cosine of the angle uniform in the range [-1,1] (*cosine*). *cosine* has the advantage of eliminating the *acos()* function call, which speeds up the compute by 2-3x, and it is also preferred on physical grounds, because the for uniformly distributed particles in 3D, the angular probability density w.r.t dtheta is sin(theta)/2, while for d(cos(theta)), it is 1/2, Regardless of which ordinate is chosen, the first column of ADF values is normalized w.r.t. the range of that ordinate, so that the integral is 1.

The simplest way to output the results of the compute adf calculation to a file is to use the *fix ave/time* command, for example:

```
compute myADF all adf 32 2 2 2 0.5 3.5 0.5 3.5
fix 1 all ave/time 100 1 100 c_myADF[*] file tmp.adf mode vector
```

3.2.4 Output info

This compute calculates a global array with the number of rows = *Nbins* and the number of columns = $1 + 2 \times$ *Ntriples*, where *Ntriples* is the number of I,J,K triples specified. The first column has the bin coordinate (angle-related ordinate at midpoint of bin). Each subsequent column has the two ADF values for a specific set of (*itypeN*,*jtypeN*,*ktypeN*) interactions, as described above. These values can be used by any command that uses a global values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

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

The first column of array values is the angle-related ordinate, either the angle in degrees or radians, or the cosine of the angle. Each subsequent pair of columns gives the first and second kinds of ADF for a specific set of (*itypeN*,*jtypeN*,*ktypeN*). The values in the first ADF column are normalized numbers ≥ 0.0, whose integral w.r.t. the ordinate is 1, i.e. the first ADF is a normalized probability distribution. The values in the second ADF column are also numbers ≥ 0.0. They are the cumulative density distribution of angles per atom. By definition, this ADF is monotonically increasing from zero to a maximum value equal to the average total number of angles per atom satisfying the ADF criteria.

3.2.5 Restrictions

This compute is part of the EXTRA-COMPUTE package. It is only enabled if LAMMPS was built with that package. See the *Build package* page for more info.

By default, the ADF is not computed for distances longer than the largest force cutoff, since the neighbor list creation will only contain pairs up to that distance (plus neighbor list skin). If you use outer cutoffs larger than that, you must use *neighbor style 'bin' or 'nsq'*.

If you want an ADF for a larger outer cutoff, you can also use the *rerun* command to post-process a dump file, use *pair style zero* and set the force cutoff to be larger in the rerun script. Note that in the rerun context, the force cutoff is arbitrary and with pair style zero you are not computing any forces, and since you are not running dynamics you are not changing the model that generated the trajectory.

The ADF is not computed for neighbors outside the force cutoff, since processors (in parallel) don't know about atom coordinates for atoms further away than that distance. If you want an ADF for larger distances, you can use the *rerun* command to post-process a dump file and set the cutoff for the potential to be longer in the rerun script. Note that in the rerun context, the force cutoff is arbitrary, since you are not running dynamics and thus are not changing your model.

3.2.6 Related commands

compute rdf, *fix ave/time*, *compute_modify*

3.2.7 Default

The keyword default is ordinate = degree.

3.3 compute angle command

3.3.1 Syntax

```
compute ID group-ID angle
```

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

3.3.2 Examples

```
compute 1 all angle
```

3.3.3 Description

Define a computation that extracts the angle energy calculated by each of the angle sub-styles used in the *angle_style hybrid* command. These values are made accessible for output or further processing by other commands. The group specified for this command is ignored.

This compute is useful when using *angle_style hybrid* if you want to know the portion of the total energy contributed by one or more of the hybrid sub-styles.

3.3.4 Output info

This compute calculates a global vector of length *N*, where *N* is the number of sub_styles defined by the *angle_style hybrid* command, which can be accessed by indices 1 through *N*. These values can be used by any command that uses global scalar or vector values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

The vector values are "extensive" and will be in energy *units*.

3.3.5 Restrictions

none

3.3.6 Related commands

compute pe, *compute pair*

3.3.7 Default

none

3.4 compute angle/local command

3.4.1 Syntax

```
compute ID group-ID angle/local value1 value2 ... keyword args ...
```

- ID, group-ID are documented in *compute* command
- angle/local = style name of this compute command
- one or more values may be appended
- value = *theta* or *eng* or *v_name*

  theta = tabulate angles
  eng = tabulate angle energies
  v_name = equal-style variable with name (see below)

- zero or more keyword/args pairs may be appended
- keyword = *set*

  set args = theta name
    theta = only currently allowed arg
    name = name of variable to set with theta

3.4.2 Examples

```
compute 1 all angle/local theta
compute 1 all angle/local eng theta
compute 1 all angle/local theta v_cos set theta t
```

3.4.3 Description

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

The value *theta* is the angle for the three atoms in the interaction.

The value *eng* is the interaction energy for the angle.

The value *v_name* can be used together with the *set* keyword to compute a user-specified function of the angle theta. The *name* specified for the *v_name* value is the name of an *equal-style variable* which should evaluate a formula based on a variable which will store the angle theta. This other variable must be an *internal-style variable* defined in the input script; its initial numeric value can be anything. It must be an internal-style variable, because this command resets its value directly. The *set* keyword is used to identify the name of this other variable associated with theta.

Note that the value of theta for each angle which stored in the internal variable is in radians, not degrees.

As an example, these commands can be added to the bench/in.rhodo script to compute the cosine and cosine-squared of every angle in the system and output the statistics in various ways:

```
variable t internal 0.0
variable cos equal cos(v_t)
variable cossq equal cos(v_t)*cos(v_t)

compute 1 all property/local aatom1 aatom2 aatom3 atype
compute 2 all angle/local eng theta v_cos v_cossq set theta t
dump 1 all local 100 tmp.dump c_1[*] c_2[*]

compute 3 all reduce ave c_2[*]
thermo_style custom step temp press c_3[*]

fix 10 all ave/histo 10 10 100 -1 1 20 c_2[3] mode vector file tmp.histo
```

The *dump local* command will output the potential energy (ϕ), the angle (θ), $\cos(\theta)$, and $\cos^2(\theta)$ for every angle θ in the system. The *thermo_style* command will print the average of those quantities via the *compute reduce* command with thermo output. And the *fix ave/histo* command will histogram the $\cos(\theta)$ values and write them to a file.

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

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

Here is an example of how to do this:

```
compute 1 all property/local atype aatom1 aatom2 aatom3
compute 2 all angle/local theta eng
dump 1 all local 1000 tmp.dump index c_1[1] c_1[2] c_1[3] c_1[4] c_2[1] c_2[2]
```

3.4.4 Output info

This compute calculates a local vector or local array depending on the number of values. The length of the vector or number of rows in the array is the number of angles. If a single value is specified, a local vector is produced. If two or more values are specified, a local array is produced where the number of columns = the number of values. The vector or array can be accessed by any command that uses local values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

The output for *theta* will be in degrees. The output for *eng* will be in energy *units*.

3.4.5 Restrictions

none

3.4.6 Related commands

dump local, *compute property/local*

3.4.7 Default

none

3.5 compute angmom/chunk command

3.5.1 Syntax

```
compute ID group-ID angmom/chunk chunkID
```

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

3.5.2 Examples

```
compute 1 fluid angmom/chunk molchunk
```

3.5.3 Description

Define a computation that calculates the angular momentum of multiple chunks of atoms.

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* and *Howto chunk* doc pages for details of how chunks can be defined and examples of how they can be used to measure properties of a system.

This compute calculates the 3 components of the angular momentum vector for each chunk, due to the velocity/momentum of the individual atoms in the chunk around the center-of-mass of the chunk. The calculation includes all effects due to atoms passing through periodic boundaries.

Note that only atoms in the specified group contribute to the calculation. The *compute chunk/atom* command defines its own group; atoms will have a chunk ID = 0 if they are not in that group, signifying they are not assigned to a chunk, and will thus also not contribute to this calculation. You can specify the "all" group for this command if you simply want to include atoms with non-zero chunk IDs.

Note

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

The simplest way to output the results of the compute angmom/chunk calculation to a file is to use the *fix ave/time* command, for example:

```
compute cc1 all chunk/atom molecule
compute myChunk all angmom/chunk cc1
fix 1 all ave/time 100 1 100 c_myChunk[*] file tmp.out mode vector
```

3.5.4 Output info

This compute calculates a global array where the number of rows = the number of chunks *Nchunk* as calculated by the specified *compute chunk/atom* command. The number of columns = 3 for the three (x, y, z) components of the angular momentum for each chunk. These values can be accessed by any command that uses global array values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

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

3.5.5 Restrictions

none

3.5.6 Related commands

variable angmom() function

3.5.7 Default

none

3.6 compute ave/sphere/atom command

Accelerator Variants: *ave/sphere/atom/kk*

3.6.1 Syntax

```
compute ID group-ID ave/sphere/atom keyword values ...
```

- ID, group-ID are documented in *compute* command
- ave/sphere/atom = style name of this compute command
- one or more keyword/value pairs may be appended

  keyword = cutoff
    cutoff value = distance cutoff

3.6.2 Examples

```
compute 1 all ave/sphere/atom

compute 1 all ave/sphere/atom cutoff 5.0
comm_modify cutoff 5.0
```

3.6.3 Description

Added in version 7Jan2022.

Define a computation that calculates the local mass density and temperature for each atom based on its neighbors inside a spherical cutoff. If an atom has M neighbors, then its local mass density is calculated as the sum of its mass and its M neighbor masses, divided by the volume of the cutoff sphere (or circle in 2d). The local temperature of the atom is calculated as the temperature of the collection of $M + 1$ atoms, after subtracting the center-of-mass velocity of the $M + 1$ atoms from each of the $M + 1$ atom's velocities. This is effectively the thermal velocity of the neighborhood of the central atom, similar to *compute temp/com*.

The optional keyword *cutoff* defines the distance cutoff used when searching for neighbors. The default value is the cutoff specified by the pair style. If no pair style is defined, then a cutoff must be defined using this keyword. If the specified cutoff is larger than that of the pair_style plus neighbor skin (or no pair style is defined), the *comm_modify cutoff* option must also be set to match that of the *cutoff* keyword.

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

Note

If you have a bonded system, then the settings of *special_bonds* command can remove pairwise interactions between atoms in the same bond, angle, or dihedral. This is the default setting for the *special_bonds* command, and means those pairwise interactions do not appear in the neighbor list. Because this compute uses the neighbor list, it also means those pairs will not be included in the order parameter. This difficulty can be circumvented by writing a dump file, and using the *rerun* command to compute the order parameter for snapshots in the dump file. The rerun script can use a *special_bonds* command that includes all pairs in the neighbor list.

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.

3.6.4 Output info

This compute calculates a per-atom array with two columns: mass density in density *units* and temperature in temperature *units*.

These values can be accessed by any command that uses per-atom values from a compute as input. See the *Howto output* doc page for an overview of LAMMPS output options.

3.6.5 Restrictions

This compute is part of the EXTRA-COMPUTE package. It is only enabled if LAMMPS was built with that package. See the *Build package* page for more info.

This compute requires *neighbor styles 'bin' or 'nsq'*.

3.6.6 Related commands

comm_modify

3.6.7 Default

The option defaults are *cutoff* = pair style cutoff.

3.7 compute basal/atom command

3.7.1 Syntax

```
compute ID group-ID basal/atom
```

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

3.7.2 Examples

```
compute 1 all basal/atom
```

3.7.3 Description

Defines a computation that calculates the hexagonal close-packed "c" lattice vector for each atom in the group. It does this by calculating the normal unit vector to the basal plane for each atom. The results enable efficient identification and characterization of twins and grains in hexagonal close-packed structures.

The output of the compute is thus the 3 components of a unit vector associated with each atom. The components are set to 0.0 for atoms not in the group.

Details of the calculation are given in *(Barrett)*.

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

An example input script that uses this compute is provided in examples/PACKAGES/basal.

3.7.4 Output info

This compute calculates a per-atom array with three columns, which can be accessed by indices 1–3 by any command that uses per-atom values from a compute as input. See the *Howto output* doc page for an overview of LAMMPS output options.

The per-atom vector values are unitless since the three columns represent components of a unit vector.

3.7.5 Restrictions

This compute is part of the EXTRA-COMPUTE package. It is only enabled if LAMMPS was built with that package. See the *Build package* page for more info.

The output of this compute will be meaningless unless the atoms are on (or near) hcp lattice sites, since the calculation assumes a well-defined basal plane.

3.7.6 Related commands

compute centro/atom, *compute ackland/atom*

3.7.7 Default

none

(Barrett) Barrett, Tschopp, El Kadiri, Scripta Mat. 66, p.666 (2012).

3.8 compute body/local command

3.8.1 Syntax

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

- ID, group-ID are documented in *compute* command
- body/local = style name of this compute command
- one or more keywords may be appended
- keyword = *id* or *type* or *integer*

  id = atom ID of the body particle
  type = atom type of the body particle
  integer = 1,2,3,etc = index of fields defined by body style

3.8.2 Examples

```
compute 1 all body/local type 1 2 3
compute 1 all body/local 3 6
```

3.8.3 Description

Define a computation that calculates properties of individual body sub-particles. The number of data generated, aggregated across all processors, equals the number of body sub-particles plus the number of non-body particles in the system, modified by the group parameter as explained below. See the *Howto body* page for more details on using body particles.

The local data stored by this command is generated by looping over all the atoms. An atom will only be included if it is in the group. If the atom is a body particle, then its N sub-particles will be looped over, and it will contribute N data to the count of data. If it is not a body particle, it will contribute 1 datum.

For both body particles and non-body particles, the *id* keyword will store the ID of the particle.

For both body particles and non-body particles, the *type* keyword will store the type of the particle.

The *integer* keywords mean different things for body and non-body particles. If the atom is not a body particle, only its x, y, z coordinates can be referenced, using the *integer* keywords 1,2,3. Note that this means that if you want to access more fields than this for body particles, then you cannot include non-body particles in the group.

For a body particle, the *integer* keywords refer to fields calculated by the body style for each sub-particle. The body style, as specified by the *atom_style body*, determines how many fields exist and what they are. See the *Howto_body* doc page for details of the different styles.

Here is an example of how to output body information using the *dump local* command with this compute. If fields 1, 2, and 3 for the body sub-particles are (x, y, z) coordinates, then the dump file will be formatted similar to the output of a *dump atom or custom* command.

```
compute 1 all body/local type 1 2 3
dump 1 all local 1000 tmp.dump index c_1[1] c_1[2] c_1[3] c_1[4]
```

3.8.4 Output info

This compute calculates a local vector or local array depending on the number of keywords. The length of the vector or number of rows in the array is the number of data as described above. If a single keyword is specified, a local vector is produced. If two or more keywords are specified, a local array is produced where the number of columns = the number of keywords. The vector or array can be accessed by any command that uses local values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

The *units* for output values depend on the body style.

3.8.5 Restrictions

none

3.8.6 Related commands

dump local

3.8.7 Default

none

3.9 compute bond command

3.9.1 Syntax

```
compute ID group-ID bond
```

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

3.9.2 Examples

```
compute 1 all bond
```

3.9.3 Description

Define a computation that extracts the bond energy calculated by each of the bond sub-styles used in the *bond_style hybrid* command. These values are made accessible for output or further processing by other commands. The group specified for this command is ignored.

This compute is useful when using *bond_style hybrid* if you want to know the portion of the total energy contributed by one or more of the hybrid sub-styles.

3.9.4 Output info

This compute calculates a global vector of length N, where N is the number of sub_styles defined by the *bond_style hybrid* command, which can be accessed by indices 1 through N. These values can be used by any command that uses global scalar or vector values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

The vector values are "extensive" and will be in energy *units*.

3.9.5 Restrictions

none

3.9.6 Related commands

compute pe, *compute pair*

3.9.7 Default

none

3.10 compute bond/local command

3.10.1 Syntax

```
compute ID group-ID bond/local value1 value2 ... keyword args ...
```

- ID, group-ID are documented in *compute* command
- bond/local = style name of this compute command
- one or more values may be appended
- value = *dist* or *dx* or *dy* or *dz* or *engpot* or *force* or *fx* or *fy* or *fz* or *engvib* or *engrot* or *engtrans* or *omega* or *velvib* or *v_name* or *bN*

dist = bond distance
engpot = bond potential energy
force = bond force

dx,dy,dz = components of pairwise distance
fx,fy,fz = components of bond force
engvib = bond kinetic energy of vibration
engrot = bond kinetic energy of rotation
engtrans = bond kinetic energy of translation
omega = magnitude of bond angular velocity
velvib = vibrational velocity along the bond length
v_name = equal-style variable with name (see below)
bN = bond style specific quantities for allowed N values

- zero or more keyword/args pairs may be appended
- keyword = *set*

set args = dist name
  dist = only currently allowed arg
  name = name of variable to set with distance (dist)

3.10.2 Examples

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

compute 1 all bond/local dist fx fy fz b1 b2

compute 1 all bond/local dist v_distsq set dist d
```

3.10.3 Description

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

All these properties are computed for the pair of atoms in a bond, whether the two atoms represent a simple diatomic molecule, or are part of some larger molecule.

The value *dist* is the current length of the bond. The values *dx*, *dy*, and *dz* are the xyz components of the *distance* between the pair of atoms. This value is always the distance from the atom of lower to the one with the higher id.

The value *engpot* is the potential energy for the bond, based on the current separation of the pair of atoms in the bond.

The value *force* is the magnitude of the force acting between the pair of atoms in the bond.

The values *fx*, *fy*, and *fz* are the xyz components of *force* between the pair of atoms in the bond. For bond styles that apply non-central forces, such as *bond_style bpm/rotational*, these values only include the (x, y, z) components of the normal force component.

The remaining properties are all computed for motion of the two atoms relative to the center of mass (COM) velocity of the two atoms in the bond.

The value *engvib* is the vibrational kinetic energy of the two atoms in the bond, which is simply $\frac{1}{2}m_1v_1^2 + \frac{1}{2}m_2v_2^2$, where v_1 and v_2 are the magnitude of the velocity of the two atoms along the bond direction, after the COM velocity has been subtracted from each.

The value *engrot* is the rotational kinetic energy of the two atoms in the bond, which is simply $\frac{1}{2}m_1v_1^2 + \frac{1}{2}m_2v_2^2$, where v_1 and v_2 are the magnitude of the velocity of the two atoms perpendicular to the bond direction, after the COM velocity has been subtracted from each.

The value *engtrans* is the translational kinetic energy associated with the motion of the COM of the system itself, namely $\frac{1}{2}(m_1 + m_2)V_{\rm cm}^2$, where *Vcm* = magnitude of the velocity of the COM.

Note that these three kinetic energy terms are simply a partitioning of the summed kinetic energy of the two atoms themselves. That is, the total kinetic energy is $\frac{1}{2}m_1v_1^2 + \frac{1}{2}m_2v_2^2$ = engvib + engrot + engtrans, where v_1 and v_2 are the magnitude of the velocities of the two atoms, without any adjustment for the COM velocity.

The value *omega* is the magnitude of the angular velocity of the two atoms around their COM position.

The value *velvib* is the magnitude of the relative velocity of the two atoms in the bond towards each other. A negative value means the two atoms are moving toward each other; a positive value means they are moving apart.

The value *v_name* can be used together with the *set* keyword to compute a user-specified function of the bond distance. The *name* specified for the *v_name* value is the name of an *equal-style variable* which should evaluate a formula based

on a variable which will store the bond distance. This other variable must be an *internal-style variable* defined in the input script; its initial numeric value can be anything. It must be an internal-style variable, because this command resets its value directly. The *set* keyword is used to identify the name of this other variable associated with theta.

As an example, these commands can be added to the bench/in.rhodo script to compute the length^2 of every bond in the system and output the statistics in various ways:

```
variable d internal 0.0
variable dsq equal v_d*v_d

compute 1 all property/local batom1 batom2 btype
compute 2 all bond/local engpot dist v_dsq set dist d
dump 1 all local 100 tmp.dump c_1[*] c_2[*]

compute 3 all reduce ave c_2[*]
thermo_style custom step temp press c_3[*]

fix 10 all ave/histo 10 10 100 0 6 20 c_2[3] mode vector file tmp.histo
```

The *dump local* command will output the energy, length, and length^2 for every bond in the system. The *thermo_style* command will print the average of those quantities via the *compute reduce* command with thermo output, and the *fix ave/histo* command will histogram the length^2 values and write them to a file.

A bond style may define additional bond quantities which can be accessed as *b1* to *bN*, where N is defined by the bond style. Most bond styles do not define any additional quantities, so N = 0. An example of ones that do are the *BPM bond styles* which store the reference state between two particles. See individual bond styles for details.

When using *bN* with bond style *hybrid*, the output will be the Nth quantity from the sub-style that computes the bonded interaction (based on bond type). If that sub-style does not define a *bN*, the output will be 0.0. The maximum allowed N is the maximum number of quantities provided by any sub-style.

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

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

Here is an example of how to do this:

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

3.10.4 Output info

This compute calculates a local vector or local array depending on the number of values. The length of the vector or number of rows in the array is the number of bonds. If a single value is specified, a local vector is produced. If two or more values are specified, a local array is produced where the number of columns = the number of values. The vector or array can be accessed by any command that uses local values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

The output for *dist* will be in distance *units*. The output for *velvib* will be in velocity *units*. The output for *omega* will be in velocity/distance *units*. The output for *engtrans*, *engvib*, *engrot*, and *engpot* will be in energy *units*. The output for *force* will be in force *units*.

3.10.5 Restrictions

none

3.10.6 Related commands

dump local, *compute property/local*

3.10.7 Default

none

3.11 compute born/matrix command

3.11.1 Syntax

```
compute ID group-ID born/matrix keyword value ...
```

- ID, group-ID are documented in *compute* command
- born/matrix = style name of this compute command
- zero or more keywords or keyword/value pairs may be appended

```
keyword = numdiff or pair or bond or angle or dihedral or improper
  numdiff values = delta virial-ID
    delta = magnitude of strain (dimensionless)
    virial-ID = ID of pressure compute for virial (string)
    (numdiff cannot be used with any other keyword)
  pair = compute pair-wise contributions
  bond = compute bonding contributions
  angle = compute angle contributions
  dihedral = compute dihedral contributions
  improper = compute improper contributions
```

3.11.2 Examples

```
compute 1 all born/matrix
compute 1 all born/matrix bond angle
compute 1 all born/matrix numdiff 1.0e-4 myvirial
```

3.11.3 Description

Added in version 4May2022.

Define a compute that calculates $\frac{\partial^2 U}{\partial \varepsilon_i \partial \varepsilon_j}$, the second derivatives of the potential energy U with respect to the strain tensor ε elements. These values are related to:

$$C^B_{i,j} = \frac{1}{V} \frac{\partial^2 U}{\partial \varepsilon_i \partial \varepsilon_j}$$

also called the Born term of elastic constants in the stress-stress fluctuation formalism. This quantity can be used to compute the elastic constant tensor. Using the symmetric Voigt notation, the elastic constant tensor can be written as a 6x6 symmetric matrix:

$$C_{i,j} = \langle C^B_{i,j} \rangle + \frac{V}{k_B T} \left(\langle \sigma_i \sigma_j \rangle \ - \langle \sigma_i \rangle \langle \sigma_j \rangle \right) + \frac{N k_B T}{V} \left(\delta_{i,j} + (\delta_{1,i} + \delta_{2,i} + \delta_{3,i}) \ * (\delta_{1,j} + \delta_{2,j} + \delta_{3,j}) \right)$$

In the above expression, σ stands for the virial stress tensor, δ is the Kronecker delta and the usual notation apply for the number of particle, the temperature and volume respectively N, T and V. k_B is the Boltzmann constant.

The Born term is a symmetric 6x6 matrix, as is the matrix of second derivatives of potential energy w.r.t strain, whose 21 independent elements are output in this order:

$$\begin{bmatrix} C_1 & C_7 & C_8 & C_9 & C_{10} & C_{11} \\ C_7 & C_2 & C_{12} & C_{13} & C_{14} & C_{15} \\ \vdots & C_{12} & C_3 & C_{16} & C_{17} & C_{18} \\ \vdots & C_{13} & C_{16} & C_4 & C_{19} & C_{20} \\ \vdots & \vdots & \vdots & C_{19} & C_5 & C_{21} \\ \vdots & \vdots & \vdots & \vdots & C_{21} & C_6 \end{bmatrix}$$

in this matrix the indices of C_k value are the corresponding element k in the global vector output by this compute. Each term comes from the sum of the derivatives of every contribution to the potential energy in the system as explained in *(VanWorkum)*.

The output can be accessed using usual Lammps routines:

```
compute 1 all born/matrix
compute 2 all pressure NULL virial
variable S1 equal -c_2[1]
variable S2 equal -c_2[2]
variable S3 equal -c_2[3]
variable S4 equal -c_2[4]
variable S5 equal -c_2[5]
variable S6 equal -c_2[6]
fix 1 all ave/time 1 1 1 v_S1 v_S2 v_S3 v_S4 v_S5 v_S6 c_1[*] file born.out
```

In this example, the file *born.out* will contain the information needed to compute the first and second terms of the elastic constant matrix in a post processing procedure. The other required quantities can be accessed using any other *LAMMPS* usual method. Several examples of this method are provided in the examples/ELASTIC_T/BORN_MATRIX directory described on the *Examples* doc page.

NOTE: In the above $C_{i,j}$ computation, the fluctuation term involving the virial stress tensor σ is the covariance between each elements. In a solid the stress fluctuations can vary rapidly, while average fluctuations can be slow to converge. A detailed analysis of the convergence rate of all the terms in the elastic tensor is provided in the paper by Clavier et al. *(Clavier)*.

Two different computation methods for the Born matrix are implemented in this compute and are mutually exclusive.

The first one is a direct computation from the analytical formula from the different terms of the potential used for the simulations *(VanWorkum)*. However, the implementation of such derivations must be done for every potential form. This has not been done yet and can be very complicated for complex potentials. At the moment a warning message is displayed for every term that is not supporting the compute at the moment. This method is the default for now.

The second method uses finite differences of energy to numerically approximate the second derivatives *(Zhen)*. This is useful when using interaction styles for which the analytical second derivatives have not been implemented. In this cases, the compute 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 stress virial tensor of the entire system is calculated after each. The difference in these two virials divided by two times *delta*, approximates the corresponding components of the second derivative, 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 keyword requires the additional arguments *delta* and *virial-ID*. *delta* gives the size of the applied strains. *virial-ID* gives the ID string of the pressure compute that provides the virial stress tensor, requiring that it use the virial keyword e.g.

```
compute myvirial all pressure NULL virial
compute 1 all born/matrix numdiff 1.0e-4 myvirial
```

Output info:

This compute calculates a global vector with 21 values that are the second derivatives of the potential energy with respect to strain. The values are in energy units. The values are ordered as explained above. These values can be used by any command that uses global values from a compute as input. See the *Howto output* doc page for an overview of LAMMPS output options.

The array values calculated by this compute are all “extensive”.

3.11.4 Restrictions

This compute is part of the EXTRA-COMPUTE package. It is only enabled if LAMMPS was built with that package. See the *Build package* page for more info. LAMMPS was built with that package. See the *Build package* page for more info.

The Born term can be decomposed as a product of two terms. The first one is a general term which depends on the configuration. The second one is specific to every interaction composing your force field (non-bonded, bonds, angle, ...). Currently not all LAMMPS interaction styles implement the *born_matrix* method giving first and second order derivatives and LAMMPS will exit with an error if this compute is used with such interactions unless the *numdiff* option is also used. The *numdiff* option cannot be used with any other keyword. In this situation, LAMMPS will also exit with an error.

3.11.5 Default

none

(Van Workum) K. Van Workum et al., J. Chem. Phys. 125 144506 (2006)

(Clavier) G. Clavier, N. Desbiens, E. Bourasseau, V. Lachet, N. Brusselle-Dupend and B. Rousseau, Mol Sim, 43, 1413 (2017).

(Zhen) Y. Zhen, C. Chu, Computer Physics Communications 183(2012)261-265

3.12 compute centro/atom command

3.12.1 Syntax

```
compute ID group-ID centro/atom lattice keyword value ...
```

- ID, group-ID are documented in *compute* command
- centro/atom = style name of this compute command
- lattice = *fcc* or *bcc* or N = # of neighbors per atom to include
- zero or more keyword/value pairs may be appended
- keyword = *axes*

  axes value = no or yes
    no = do not calculate 3 symmetry axes
    yes = calculate 3 symmetry axes

3.12.2 Examples

```
compute 1 all centro/atom fcc

compute 1 all centro/atom 8
```

3.12.3 Description

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

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

This parameter is computed using the following formula from *(Kelchner)*

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

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

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

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

If the keyword *axes* has the setting *yes*, then this compute also estimates three symmetry axes for each atom's local neighborhood. The first two of these are the vectors joining the two pairs of neighbor atoms with smallest contributions to the centrosymmetry parameter, i.e. the two most symmetric pairs of atoms. The third vector is normal to the first two by the right-hand rule. All three vectors are normalized to unit length. For FCC crystals, the first two vectors will lie along a $\langle 110 \rangle$ direction, while the third vector will lie along either a $\langle 100 \rangle$ or $\langle 111 \rangle$ direction. For HCP crystals, the first two vectors will lie along $\langle 1000 \rangle$ directions, while the third vector will lie along $\langle 0001 \rangle$. This provides a simple way to measure local orientation in HCP structures. In general, the *axes* keyword can be used to estimate the orientation of symmetry axes in the neighborhood of any atom.

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

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

3.12.4 Output info

By default, this compute calculates the centrosymmetry value for each atom as a per-atom vector, which can be accessed by any command that uses per-atom values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

If the *axes* keyword setting is *yes*, then a per-atom array is calculated. The first column is the centrosymmetry parameter. The next three columns are the x, y, and z components of the first symmetry axis, followed by the second, and third symmetry axes in columns 5–7 and 8–10.

The centrosymmetry values are unitless values ≥ 0.0. Their magnitude depends on the lattice style due to the number of contributing neighbor pairs in the summation in the formula above. And it depends on the local defects surrounding the central atom, as described above. For the *axes yes* case, the vector components are also unitless, since they represent spatial directions.

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

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

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

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

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

3.12.5 Restrictions

none

3.12.6 Related commands

compute cna/atom

3.12.7 Default

The default value for the optional keyword is axes = no.

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

3.13 compute chunk/atom command

3.13.1 Syntax

```
compute ID group-ID chunk/atom style args keyword values ...
```

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

  style = bin/1d or bin/2d or bin/3d or bin/sphere or bin/cylinder or type or molecule or c_ID, c_ID[I], f_ID, f_ID[I], v_name
    bin/1d args = dim origin delta
      dim = x or y or z
      origin = lower or center or upper or coordinate value (distance units)
      delta = thickness of spatial bins in dim (distance units)
    bin/2d args = dim origin delta dim origin delta
      dim = x or y or z
      origin = lower or center or upper or coordinate value (distance units)
      delta = thickness of spatial bins in dim (distance units)
    bin/3d args = dim origin delta dim origin delta dim origin delta
      dim = x or y or z
      origin = lower or center or upper or coordinate value (distance units)
      delta = thickness of spatial bins in dim (distance units)
    bin/sphere args = xorig yorig zorig rmin rmax nsbin
      xorig,yorig,zorig = center point of sphere
      srmin,srmax = bin from sphere radius rmin to rmax
      nsbin = # of spherical shell bins between rmin and rmax
    bin/cylinder args = dim origin delta c1 c2 rmin rmax ncbin
      dim = x or y or z = axis of cylinder axis
      origin = lower or center or upper or coordinate value (distance units)
      delta = thickness of spatial bins in dim (distance units)
      c1,c2 = coords of cylinder axis in other 2 dimensions (distance units)
      crmin,crmax = bin from cylinder radius rmin to rmax (distance units)
      ncbin = # of concentric circle bins between rmin and rmax
    type args = none
    molecule args = none
    c_ID, c_ID[I], f_ID, f_ID[I], v_name args = none
      c_ID = per-atom vector calculated by a compute with ID
      c_ID[I] = Ith column of per-atom array calculated by a compute with ID
      f_ID = per-atom vector calculated by a fix with ID
      f_ID[I] = Ith column of per-atom array calculated by a fix with ID
      v_name = per-atom vector calculated by an atom-style variable with name

- zero or more keyword/values pairs may be appended
- keyword = *region* or *nchunk* or *limit* or *ids* or *compress* or *discard* or *bound* or *pbc* or *units*

  region value = region-ID
    region-ID = ID of region atoms must be in to be part of a chunk
  nchunk value = once or every
    once = only compute the number of chunks once
    every = re-compute the number of chunks whenever invoked
  limit values = 0 or Nc max or Nc exact
    0 = no limit on the number of chunks

```
  Nc max = limit number of chunks to be <= Nc
  Nc exact = set number of chunks to exactly Nc
ids value = once or nfreq or every
  once = assign chunk IDs to atoms only once, they persist thereafter
  nfreq = assign chunk IDs to atoms only once every Nfreq steps (if invoked by fix ave/chunk which
 ↪sets Nfreq)
  every = assign chunk IDs to atoms whenever invoked
compress value = yes or no
  yes = compress chunk IDs to eliminate IDs with no atoms
  no = do not compress chunk IDs even if some IDs have no atoms
discard value = yes or no or mixed
  yes = discard atoms with out-of-range chunk IDs by assigning a chunk ID = 0
  no = keep atoms with out-of-range chunk IDs by assigning a valid chunk ID
  mixed = keep or discard such atoms according to spatial binning rule
bound values = x/y/z lo hi
  x/y/z = x or y or z to bound spatial bins in this dimension
  lo = lower or coordinate value (distance units)
  hi = upper or coordinate value (distance units)
pbc value = no or yes
  yes = use periodic distance for bin/sphere and bin/cylinder styles
units value = box or lattice or reduced
```

3.13.2 Examples

```
compute 1 all chunk/atom type
compute 1 all chunk/atom bin/1d z lower 0.02 units reduced
compute 1 all chunk/atom bin/2d z lower 1.0 y 0.0 2.5
compute 1 all chunk/atom molecule region sphere nchunk once ids once compress yes
compute 1 all chunk/atom bin/sphere 5 5 5 2.0 5.0 5 discard yes
compute 1 all chunk/atom bin/cylinder z lower 2 10 10 2.0 5.0 3 discard yes
compute 1 all chunk/atom c_cluster
```

3.13.3 Description

Define a computation that calculates an integer chunk ID from 1 to Nchunk for each atom in the group. Values of chunk IDs are determined by the *style* of chunk, which can be based on atom type or molecule ID or spatial binning or a per-atom property or value calculated by another *compute*, *fix*, or *atom-style variable*. Per-atom chunk IDs can be used by other computes with "chunk" in their style name, such as *compute com/chunk* or *compute msd/chunk*. Or they can be used by the *fix ave/chunk* command to sum and time average a variety of per-atom properties over the atoms in each chunk. Or they can simply be accessed by any command that uses per-atom values from a compute as input, as discussed on the *Howto output* doc page.

See the *Howto chunk* page for an overview of how this compute can be used with a variety of other commands to tabulate properties of a simulation. The page gives several examples of input script commands that can be used to calculate interesting properties.

Conceptually it is important to realize that this compute does two simple things. First, it sets the value of *Nchunk* = the number of chunks, which can be a constant value or change over time. Second, it assigns each atom to a chunk via a chunk ID. Chunk IDs range from 1 to *Nchunk* inclusive; some chunks may have no atoms assigned to them. Atoms that do not belong to any chunk are assigned a value of 0. Note that the two operations are not always performed together. For example, spatial bins can be setup once (which sets *Nchunk*), and atoms assigned to those bins many times thereafter (setting their chunk IDs).

All other commands in LAMMPS that use chunk IDs assume there are *Nchunk* number of chunks, and that every atom is assigned to one of those chunks, or not assigned to any chunk.

There are many options for specifying for how and when *Nchunk* is calculated, and how and when chunk IDs are assigned to atoms. The details depend on the chunk *style* and its *args*, as well as optional keyword settings. They can also depend on whether a *fix ave/chunk* command is using this compute, since that command requires *Nchunk* to remain static across windows of timesteps it specifies, while it accumulates per-chunk averages.

The details are described below.

The different chunk styles operate as follows. For each style, how it calculates *Nchunk* and assigns chunk IDs to atoms is explained. Note that using the optional keywords can change both of those actions, as described further below where the keywords are discussed.

The *binning* styles perform a spatial binning of atoms, and assign an atom the chunk ID corresponding to the bin number it is in. *Nchunk* is set to the number of bins, which can change if the simulation box size changes. This also depends on the setting of the *units* keyword (e.g., for *reduced* units the number of chunks may not change even if the box size does).

The *bin/1d*, *bin/2d*, and *bin/3d* styles define bins as 1d layers (slabs), 2d pencils, or 3d boxes. The *dim*, *origin*, and *delta* settings are specified 1, 2, or 3 times. For 2d or 3d bins, there is no restriction on specifying dim = x before dim = y or z, or dim = y before dim = z. Bins in a particular *dim* have a bin size in that dimension given by *delta*. In each dimension, bins are defined relative to a specified *origin*, which may be the lower/upper edge of the simulation box (in that dimension), or its center point, or a specified coordinate value. Starting at the origin, sufficient bins are created in both directions to completely span the simulation box or the bounds specified by the optional *bounds* keyword.

For orthogonal simulation boxes, the bins are layers, pencils, or boxes aligned with the xyz coordinate axes. For triclinic (non-orthogonal) simulation boxes, the bin faces are parallel to the tilted faces of the simulation box. See the *Howto triclinic* page for a discussion of the geometry of triclinic boxes in LAMMPS. As described there, a tilted simulation box has edge vectors $\vec{a}$, $\vec{b}$, and $\vec{c}$. In that nomenclature, bins in the x dimension have faces with normals in the $\vec{b} \times \vec{c}$ direction, bins in y have faces normal to the $\vec{a} \times \vec{c}$ direction, and bins in z have faces normal to the $\vec{a} \times \vec{b}$ direction. Note that in order to define the size and position of these bins in an unambiguous fashion, the *units* option must be set to *reduced* when using a triclinic simulation box, as noted below.

The meaning of *origin* and *delta* for triclinic boxes is as follows. Consider a triclinic box with bins that are 1d layers or slabs in the x dimension. No matter how the box is tilted, an *origin* of 0.0 means start layers at the lower $\vec{b} \times \vec{c}$ plane of the simulation box and an *origin* of 1.0 means to start layers at the upper $\vec{b} \times \vec{c}$ face of the box. A *delta* value of 0.1 in *reduced* units means there will be 10 layers from 0.0 to 1.0, regardless of the current size or shape of the simulation box.

The *bin/sphere* style defines a set of spherical shell bins around the origin (*xorig*,*yorig*,*zorig*), using *nsbin* bins with radii equally spaced between *srmin* and *srmax*. This is effectively a 1d vector of bins. For example, if *srmin* = 1.0 and *srmax* = 10.0 and *nsbin* = 9, then the first bin spans $1.0 < r < 2.0$, and the last bin spans $9.0 < r < 10.0$. The geometry of the bins is the same whether the simulation box is orthogonal or triclinic (i.e., the spherical shells are not tilted or scaled differently in different dimensions to transform them into ellipsoidal shells).

The *bin/cylinder* style defines bins for a cylinder oriented along the axis *dim* with the axis coordinates in the other two radial dimensions at (*c1*,*c2*). For dim = x, $c_1/c_2 = y/z$; for dim = y, $c_1/c_2 = x/z$; for dim = z, $c_1/c_2 = x/y$. This is effectively a 2d array of bins. The first dimension is along the cylinder axis, the second dimension is radially outward from the cylinder axis. The bin size and positions along the cylinder axis are specified by the *origin* and *delta* values, the same as for the *bin/1d*, *bin/2d*, and *bin/3d* styles. There are *ncbin* concentric circle bins in the radial direction from the cylinder axis with radii equally spaced between *crmin* and *crmax*. For example, if *crmin* = 1.0 and *crmax* = 10.0 and *ncbin* = 9, then the first bin spans $1.0 < r < 2.0$ and the last bin spans $9.0 < r < 10.0$. The geometry of the bins in the radial dimensions is the same whether the simulation box is orthogonal or triclinic (i.e., the concentric circles are not tilted or scaled differently in the two different dimensions to transform them into ellipses).

The created bins (and hence the chunk IDs) are numbered consecutively from 1 to the number of bins = *Nchunk*. For *bin2d* and *bin3d*, the numbering varies most rapidly in the first dimension (which could be x, y, or z), next rapidly in the second dimension, and most slowly in the third dimension. For *bin/sphere*, the bin with smallest radii is chunk 1 and the bin with largest radii is chunk Nchunk = *ncbin*. For *bin/cylinder*, the numbering varies most rapidly in the dimension along the cylinder axis and most slowly in the radial direction.

Each time this compute is invoked, each atom is mapped to a bin based on its current position. Note that between reneighboring timesteps, atoms can move outside the current simulation box. If the box is periodic (in that dimension) the atom is remapping into the periodic box for purposes of binning. If the box in not periodic, the atom may have moved outside the bounds of all bins. If an atom is not inside any bin, the *discard* keyword is used to determine how a chunk ID is assigned to the atom.

The *type* style uses the atom type as the chunk ID. *Nchunk* is set to the number of atom types defined for the simulation (e.g., via the *create_box* or *read_data* commands).

The *molecule* style uses the molecule ID of each atom as its chunk ID. *Nchunk* is set to the largest chunk ID. Note that this excludes molecule IDs for atoms which are not in the specified group or optional region.

There is no requirement that all atoms in a particular molecule are assigned the same chunk ID (zero or non-zero), though you probably want that to be the case, if you wish to compute a per-molecule property. LAMMPS will issue a warning if that is not the case, but only the first time that *Nchunk* is calculated.

Note that atoms with a molecule ID = 0, which may be non-molecular solvent atoms, have an out-of-range chunk ID. These atoms are discarded (not assigned to any chunk) or assigned to *Nchunk*, depending on the value of the *discard* keyword.

The *compute/fix/variable* styles set the chunk ID of each atom based on a quantity calculated and stored by a compute, fix, or variable. In each case, it must be a per-atom quantity. In each case the referenced floating point values are converted to an integer chunk ID as follows. The floating point value is truncated (rounded down) to an integer value. If the integer value is ≤ 0, then a chunk ID of 0 is assigned to the atom. If the integer value is > 0, it becomes the chunk ID to the atom. *Nchunk* is set to the largest chunk ID. Note that this excludes atoms which are not in the specified group or optional region.

If the style 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*.

If the style 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 the timestep on which this compute accesses the fix, else an error results. Users can also write code for their own fix styles and *add them to LAMMPS*.

If a value begins with "v_", a variable name for an *atom* or *atomfile* style *variable* 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 treat as a chunk ID.

Normally, *Nchunk* = the number of chunks, is re-calculated every time this fix is invoked, though the value may or may not change. As explained below, the *nchunk* keyword can be set to *once* which means *Nchunk* will never change.

If a *fix ave/chunk* command uses this compute, it can also turn off the re-calculation of *Nchunk* for one or more windows of timesteps. The extent of the windows, during which Nchunk is held constant, are determined by the *Nevery*, *Nrepeat*, *Nfreq* values and the *ave* keyword setting that are used by the *fix ave/chunk* command.

Specifically, if *ave* = *one*, then for each span of *Nfreq* timesteps, *Nchunk* is held constant between the first timestep when averaging is done (within the Nfreq-length window), and the last timestep when averaging is done (multiple of Nfreq). If *ave* = *running* or *window*, then *Nchunk* is held constant forever, starting on the first timestep when the *fix ave/chunk* command invokes this compute.

Note that multiple *fix ave/chunk* commands can use the same compute chunk/atom compute. However, the time windows they induce for holding *Nchunk* constant must be identical, else an error will be generated.

The various optional keywords operate as follows. Note that some of them function differently or are ignored by different chunk styles. Some of them also have different default values, depending on the chunk style, as listed below.

The *region* keyword applies to all chunk styles. If used, an atom must be in both the specified group and the specified geometric *region* to be assigned to a chunk.

The *nchunk* keyword applies to all chunk styles. It specifies how often *Nchunk* is recalculated, which in turn can affect the chunk IDs assigned to individual atoms.

If *nchunk* is set to *once*, then *Nchunk* is only calculated once, the first time this compute is invoked. If *nchunk* is set to *every*, then *Nchunk* is re-calculated every time the compute is invoked. Note that, as described above, the use of this compute by the *fix ave/chunk* command can override the *every* setting.

The default values for *nchunk* are listed below and depend on the chunk style and other system and keyword settings. They attempt to represent typical use cases for the various chunk styles. The *nchunk* value can always be set explicitly if desired.

The *limit* keyword can be used to limit the calculated value of *Nchunk* = the number of chunks. The limit is applied each time *Nchunk* is calculated, which also limits the chunk IDs assigned to any atom. The *limit* keyword is used by all chunk styles except the *binning* styles, which ignore it. This is because the number of bins can be tailored using the *bound* keyword (described below) which effectively limits the size of *Nchunk*.

If *limit* is set to *Nc* = 0, then no limit is imposed on *Nchunk*, though the *compress* keyword can still be used to reduce *Nchunk*, as described below.

If $Nc > 0$, then the effect of the *limit* keyword depends on whether the *compress* keyword is also used with a setting of *yes*, and whether the *compress* keyword is specified before the *limit* keyword or after.

In all cases, *Nchunk* is first calculated in the usual way for each chunk style, as described above.

First, here is what occurs if *compress yes* is not set. If *limit* is set to *Nc max*, then *Nchunk* is reset to the smaller of *Nchunk* and *Nc*. If *limit* is set to *Nc exact*, then *Nchunk* is reset to *Nc*, whether the original *Nchunk* was larger or smaller than *Nc*. If *Nchunk* shrank due to the *limit* setting, then atom chunk IDs $>$ *Nchunk* will be reset to 0 or *Nchunk*, depending on the setting of the *discard* keyword. If *Nchunk* grew, there will simply be some chunks with no atoms assigned to them.

If *compress yes* is set, and the *compress* keyword comes before the *limit* keyword, the compression operation is performed first, as described below, which resets *Nchunk*. The *limit* keyword is then applied to the new *Nchunk* value, exactly as described in the preceding paragraph. Note that in this case, all atoms will end up with chunk IDs $\leq$ *Nc*, but their original values (e.g., molecule ID or compute/fix/variable) may have been $>$ *Nc*, because of the compression operation.

If *compress yes* is set, and the *compress* keyword comes after the *limit* keyword, then the *limit* value of *Nc* is applied first to the uncompressed value of *Nchunk*, but only if *Nc* $<$ *Nchunk* (whether *Nc max* or *Nc exact* is used). This effectively

means all atoms with chunk IDs $>$ *Nc* have their chunk IDs reset to 0 or *Nc*, depending on the setting of the *discard* keyword. The compression operation is then performed, which may shrink *Nchunk* further. If the new *Nchunk* $<$ *Nc* and *limit* = *Nc exact* is specified, then *Nchunk* is reset to *Nc*, which results in extra chunks with no atoms assigned to them. Note that in this case, all atoms will end up with chunk IDs $\leq$ *Nc*, and their original values (e.g., molecule ID or compute/fix/variable value) will also have been $\leq$ *Nc*.

The *ids* keyword applies to all chunk styles. If the setting is *once* then the chunk IDs assigned to atoms the first time this compute is invoked will be permanent, and never be re-computed.

If the setting is *nfreq* and if a *fix ave/chunk* command is using this compute, then in each of the *Nchunk* = constant time windows (discussed above), the chunk ID's assigned to atoms on the first step of the time window will persist until the end of the time window.

If the setting is *every*, which is the default, then chunk IDs are re-calculated on any timestep this compute is invoked.

Note

If you want the persistent chunk-IDs calculated by this compute to be continuous when running from a *restart file*, then you should use the same ID for this compute, as in the original run. This is so that the fix this compute creates to store per-atom quantities will also have the same ID, and thus be initialized correctly with chunk IDs from the restart file.

The *compress* keyword applies to all chunk styles and affects how *Nchunk* is calculated, which in turn affects the chunk IDs assigned to each atom. It is useful for converting a "sparse" set of chunk IDs (with many IDs that have no atoms assigned to them), into a "dense" set of IDs, where every chunk has one or more atoms assigned to it.

Two possible use cases are as follows. If a large simulation box is mostly empty space, then the *binning* style may produce many bins with no atoms. If *compress* is set to *yes*, only bins with atoms will be contribute to *Nchunk*. Likewise, the *molecule* or *compute/fix/variable* styles may produce large *Nchunk* values. For example, the *compute cluster/atom* command assigns every atom an atom ID for one of the atoms it is clustered with. For a million-atom system with 5 clusters, there would only be 5 unique chunk IDs, but the largest chunk ID might be 1 million, resulting in *Nchunk* = 1 million. If *compress* is set to *yes*, *Nchunk* will be reset to 5.

If *compress* is set to *no*, which is the default, no compression is done. If it is set to *yes*, all chunk IDs with no atoms are removed from the list of chunk IDs, and the list is sorted. The remaining chunk IDs are renumbered from 1 to *Nchunk* where *Nchunk* is the new length of the list. The chunk IDs assigned to each atom reflect the new renumbering from 1 to *Nchunk*.

The original chunk IDs (before renumbering) can be accessed by the *compute property/chunk* command and its *id* keyword, or by the *fix ave/chunk* command which outputs the original IDs as one of the columns in its global output array. For example, using the "compute cluster/atom" command discussed above, the original 5 unique chunk IDs might be atom IDs (27,4982,58374,857838,1000000). After compression, these will be renumbered to (1,2,3,4,5). The original values (27,...,1000000) can be output to a file by the *fix ave/chunk* command, or by using the *fix ave/time* command in conjunction with the *compute property/chunk* command.

Note

The compression operation requires global communication across all processors to share their chunk ID values. It can require large memory on every processor to store them, even after they are compressed, if there are a large number of unique chunk IDs with atoms assigned to them. It uses a STL map to find unique chunk IDs and store them in sorted order. Each time an atom is assigned a compressed chunk ID, it must access the STL map. All of this means that compression can be expensive, both in memory and CPU time. The use of the *limit* keyword in

conjunction with the *compress* keyword can affect these costs, depending on which keyword is used first. So use this option with care.

The *discard* keyword applies to all chunk styles. It affects what chunk IDs are assigned to atoms that do not match one of the valid chunk IDs from 1 to *Nchunk*. Note that it does not apply to atoms that are not in the specified group or optionally specified region. Those atoms are always assigned a chunk ID = 0.

If the calculated chunk ID for an atom is not within the range 1 to *Nchunk* then it is a "discard" atom. Note that *Nchunk* may have been shrunk by the *limit* keyword. Or the *compress* keyword may have eliminated chunk IDs that were valid before the compression took place, and are now not in the compressed list. Also note that for the *molecule* chunk style, if new molecules are added to the system, their chunk IDs may exceed a previously calculated *Nchunk*. Likewise, evaluation of a compute/fix/variable on a later timestep may return chunk IDs that are invalid for the previously calculated *Nchunk*.

All the chunk styles except the *binning* styles, must use *discard* set to either *yes* or *no*. If *discard* is set to *yes*, which is the default, then every "discard" atom has its chunk ID set to 0. If *discard* is set to *no*, every "discard" atom has its chunk ID set to *Nchunk*. I.e. it becomes part of the last chunk.

The *binning* styles use the *discard* keyword to decide whether to discard atoms outside the spatial domain covered by bins, or to assign them to the bin they are nearest to.

For the *bin/1d*, *bin/2d*, *bin/3d* styles the details are as follows. If *discard* is set to *yes*, an out-of-domain atom will have its chunk ID set to 0. If *discard* is set to *no*, the atom will have its chunk ID set to the first or last bin in that dimension. If *discard* is set to *mixed*, which is the default, it will only have its chunk ID set to the first or last bin if bins extend to the simulation box boundary in that dimension. This is the case if the *bound* keyword settings are *lower* and *upper*, which is the default. If the *bound* keyword settings are numeric values, then the atom will have its chunk ID set to 0 if it is outside the bounds of any bin. Note that in this case, it is possible that the first or last bin extends beyond the numeric *bounds* settings, depending on the specified *origin*. If this is the case, the chunk ID of the atom is only set to 0 if it is outside the first or last bin, not if it is simply outside the numeric *bounds* setting.

For the *bin/sphere* style the details are as follows. If *discard* is set to *yes*, an out-of-domain atom will have its chunk ID set to 0. If *discard* is set to *no* or *mixed*, the atom will have its chunk ID set to the first or last bin, i.e. the innermost or outermost spherical shell. If the distance of the atom from the origin is less than *rmin*, it will be assigned to the first bin. If the distance of the atom from the origin is greater than *rmax*, it will be assigned to the last bin.

For the *bin/cylinder* style the details are as follows. If *discard* is set to *yes*, an out-of-domain atom will have its chunk ID set to 0. If *discard* is set to *no*, the atom will have its chunk ID set to the first or last bin in both the radial and axis dimensions. If *discard* is set to *mixed*, which is the default, the radial dimension is treated the same as for *discard* = no. But for the axis dimension, it will only have its chunk ID set to the first or last bin if bins extend to the simulation box boundary in the axis dimension. This is the case if the *bound* keyword settings are *lower* and *upper*, which is the default. If the *bound* keyword settings are numeric values, then the atom will have its chunk ID set to 0 if it is outside the bounds of any bin. Note that in this case, it is possible that the first or last bin extends beyond the numeric *bounds* settings, depending on the specified *origin*. If this is the case, the chunk ID of the atom is only set to 0 if it is outside the first or last bin, not if it is simply outside the numeric *bounds* setting.

If *discard* is set to *no* or *mixed*, the atom will have its chunk ID set to the first or last bin, i.e. the innermost or outermost spherical shell. If the distance of the atom from the origin is less than *rmin*, it will be assigned to the first bin. If the distance of the atom from the origin is greater than *rmax*, it will be assigned to the last bin.

The *bound* keyword only applies to the *bin/1d*, *bin/2d*, *bin/3d* styles and to the axis dimension of the *bin/cylinder* style; otherwise it is ignored. It can be used one or more times to limit the extent of bin coverage in a specified dimension, i.e. to only bin a portion of the box. If the *lo* setting is *lower* or the *hi* setting is *upper*, the bin extent in that direction extends to the box boundary. If a numeric value is used for *lo* and/or *hi*, then the bin extent in the *lo* or *hi* direction extends only to that value, which is assumed to be inside (or at least near) the simulation box boundaries, though LAMMPS does

not check for this. Note that using the *bound* keyword typically reduces the total number of bins and thus the number of chunks *Nchunk*.

The *pbc* keyword only applies to the *bin/sphere* and *bin/cylinder* styles. If set to *yes*, the distance an atom is from the sphere origin or cylinder axis is calculated in a minimum image sense with respect to periodic dimensions, when determining which bin the atom is in. I.e. if x is a periodic dimension and the distance between the atom and the sphere center in the x dimension is greater than 0.5 * simulation box length in x, then a box length is subtracted to give a distance < 0.5 * simulation box length. This allosws the sphere or cylinder center to be near a box edge, and atoms on the other side of the periodic box will still be close to the center point/axis. Note that with a setting of *yes*, the outer sphere or cylinder radius must also be <= 0.5 * simulation box length in any periodic dimension except for the cylinder axis dimension, or an error is generated.

The *units* keyword only applies to the *binning* styles; otherwise it is ignored. For the *bin/1d*, *bin/2d*, *bin/3d* styles, it determines the meaning of the distance units used for the bin sizes *delta* and for *origin* and *bounds* values if they are coordinate values. For the *bin/sphere* style it determines the meaning of the distance units used for *xorig*,*yorig*,*zorig* and the radii *srmin* and *srmax*. For the *bin/cylinder* style it determines the meaning of the distance units used for *delta*,*c1*,*c2* and the radii *crmin* and *crmax*.

For orthogonal simulation boxes, any of the 3 options may be used. For non-orthogonal (triclinic) simulation boxes, only the *reduced* option may be used.

A *box* value selects standard distance units as defined by the *units* command (e.g., 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 spacing. A *reduced* value means normalized unitless values between 0 and 1, which represent the lower and upper faces of the simulation box respectively. Thus an *origin* value of 0.5 means the center of the box in any dimension. A *delta* value of 0.1 means 10 bins span the box in that dimension.

Note that for the *bin/sphere* style, the radii *srmin* and *srmax* are scaled by the lattice spacing or reduced value of the *x* dimension.

Note that for the *bin/cylinder* style, the radii *crmin* and *crmax* are scaled by the lattice spacing or reduced value of the first dimension perpendicular to the cylinder axis (e.g., *y* for an *x*-axis cylinder, *x* for a *y*-axis cylinder, and *x* for a *z*-axis cylinder).

3.13.4 Output info

This compute calculates a per-atom vector (the chunk ID), which can be accessed by any command that uses per-atom values from a compute as input. It also calculates a global scalar (the number of chunks), which can be similarly accessed everywhere outside of a per-atom context. See the *Howto output* page for an overview of LAMMPS output options.

The per-atom vector values are unitless chunk IDs, ranging from 1 to *Nchunk* (inclusive) for atoms assigned to chunks, and 0 for atoms not belonging to a chunk. The scalar contains the value of *Nchunk*.

3.13.5 Restrictions

Even if the *nchunk* keyword is set to *once*, the chunk IDs assigned to each atom are not stored in a restart files. This means you cannot expect those assignments to persist in a restarted simulation. Instead you must re-specify this command and assign atoms to chunks when the restarted simulation begins.

3.13.6 Related commands

fix ave/chunk, *compute global/atom*

3.13.7 Default

The option defaults are as follows:

- region = none
- nchunk = every, if compress is yes, overriding other defaults listed here
- nchunk = once, for type style
- nchunk = once, for mol style if region is none
- nchunk = every, for mol style if region is set
- nchunk = once, for binning style if the simulation box size is static or units = reduced
- nchunk = every, for binning style if the simulation box size is dynamic and units is lattice or box
- nchunk = every, for compute/fix/variable style
- limit = 0
- ids = every
- compress = no
- discard = yes, for all styles except binning
- discard = mixed, for binning styles
- bound = lower and upper in all dimensions
- pbc = no
- units = lattice

3.14 compute chunk/spread/atom command

3.14.1 Syntax

```
compute ID group-ID chunk/spread/atom chunkID input1 input2 ...
```

- ID, group-ID are documented in *compute* command
- chunk/spread/atom = style name of this compute command
- chunkID = ID of *compute chunk/atom* command
- one or more inputs can be listed
- input = c_ID, c_ID[N], f_ID, f_ID[N]

```
c_ID = global vector calculated by a compute with ID
c_ID[I] = Ith column of global array calculated by a compute with ID, I can include wildcard (see␣
↪below)
f_ID = global vector calculated by a fix with ID
f_ID[I] = Ith column of global array calculated by a fix with ID, I can include wildcard (see below)
```

3.14.2 Examples

```
compute 1 all chunk/spread/atom mychunk c_com[*] c_gyration
```

3.14.3 Description

Define a calculation that "spreads" one or more per-chunk values to each atom in the chunk. This can be useful in several scenarios:

- For creating a *dump file* where each atom lists info about the chunk it is in, e.g. for post-processing purposes.
- To access chunk value in *atom-style variables* that need info about the chunk each atom is in.
- To use the *fix ave/chunk* command to spatially average per-chunk values calculated by a per-chunk compute.

Examples are given below.

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* and *Howto chunk* doc pages for details of how chunks can be defined and examples of how they can be used to measure properties of a system.

For inputs that are computes, they must be a compute that calculates per-chunk values. These are computes whose style names end in "/chunk".

For inputs that are fixes, they should be a fix that calculates per-chunk values. For example, *fix ave/chunk* or *fix ave/time* (assuming it is time-averaging per-chunk data).

For each atom, this compute accesses its chunk ID from the specified *chunkID* compute, then accesses the per-chunk value in each input. Those values are copied to this compute to become the output for that atom.

The values generated by this compute will be 0.0 for atoms not in the specified compute group *group-ID*. They will also be 0.0 if the atom is not in a chunk, as assigned by the *chunkID* compute. They will also be 0.0 if the current chunk ID for the atom is out-of-bounds with respect to the number of chunks stored by a particular input compute or fix.

Note

LAMMPS does not check that a compute or fix which calculates per-chunk values uses the same definition of chunks as this compute. It's up to you to be consistent. Likewise, for a fix input, LAMMPS does not check that it is per-chunk data. It only checks that the fix produces a global vector or array.

Each listed input is operated on independently.

If a bracketed index I is used, it 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, 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. these 2 compute chunk/spread/atom commands are equivalent, since the *compute com/chunk* command creates a per-atom array with 3 columns:

```
compute com all com/chunk mychunk
compute 10 all chunk/spread/atom mychunk c_com[*]
compute 10 all chunk/spread/atom mychunk c_com[1] c_com[2] c_com[3]
```

Here is an example of writing a dump file the with the center-of-mass (COM) for the chunk each atom is in. The commands below can be added to the bench/in.chain script.

```
compute         cmol all chunk/atom molecule
compute         com all com/chunk cmol
compute         comchunk all chunk/spread/atom cmol c_com[*]
dump            1 all custom 50 tmp.dump id mol type x y z c_comchunk[*]
dump_modify     1 sort id
```

The same per-chunk data for each atom could be used to define per-atom forces for the *fix addforce* command. In this example the forces act to pull atoms of an extended polymer chain towards its COM in an attractive manner.

```
compute         prop all property/atom xu yu zu
variable        k equal 0.1
variable        fx atom v_k*(c_comchunk[1]-c_prop[1])
variable        fy atom v_k*(c_comchunk[2]-c_prop[2])
variable        fz atom v_k*(c_comchunk[3]-c_prop[3])
fix             3 all addforce v_fx v_fy v_fz
```

Note that *compute property/atom* is used to generate unwrapped coordinates for use in the per-atom force calculation, so that the effect of periodic boundaries is accounted for properly.

Over time this applied force could shrink each polymer chain's radius of gyration in a polymer mixture simulation. Here is output from the bench/in.chain script. Thermo output is shown for 1000 steps, where the last column is the average radius of gyration over all 320 chains in the 32000 atom system:

```
compute         gyr all gyration/chunk cmol
variable        ave equal ave(c_gyr)
thermo_style    custom step etotal press v_ave

     0   22.394765    4.6721833     5.128278
   100   22.445002    4.8166709    5.0348372
   200   22.500128    4.8790392    4.9364875
   300   22.534686    4.9183766    4.8590693
   400   22.557196    4.9492211    4.7937849
   500   22.571017    4.9161853    4.7412008
   600   22.573944    5.0229708    4.6931243
   700   22.581804    5.0541301    4.6440647
   800   22.584683    4.9691734    4.6000016
   900    22.59128    5.0247538    4.5611513
  1000   22.586832      4.94697    4.5238362
```

Here is an example for using one set of chunks, defined for molecules, to compute the dipole moment vector for each chunk. E.g. for water molecules. Then spreading those values to each atom in each chunk. Then defining a second set of chunks based on spatial bins. And finally, using the *fix ave/chunk* command to calculate an average dipole moment vector per spatial bin.

```
compute         cmol all chunk/atom molecule
compute         dipole all dipole/chunk cmol
compute         spread all chunk/spread/atom cmol c_dipole[1] c_dipole[2] c_dipole[3]
compute         cspatial all chunk/atom bin/1d z lower 0.1 units reduced
fix             ave all ave/chunk 100 10 1000 cspatial c_spread[*]
```

Note that the *fix ave/chunk* command requires per-atom values as input. That is why the compute chunk/spread/atom command is used to assign per-chunk values to each atom in the chunk. If a molecule straddles bin boundaries, each of its atoms contributes in a weighted manner to the average dipole moment of the spatial bin it is in.

3.14.4 Output info

This compute calculates a per-atom vector or array, which can be accessed by any command that uses per-atom values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

The output is a per-atom vector if a single input value is specified, otherwise a per-atom array is output. The number of columns in the array is the number of inputs provided. The per-atom values for the vector or each column of the array will be in whatever *units* the corresponding input value is in.

The vector or array values are "intensive".

3.14.5 Restrictions

none

3.14.6 Related commands

compute chunk/atom, *fix ave/chunk*, *compute reduce/chunk*

3.14.7 Default

none

3.15 compute cluster/atom command

3.16 compute fragment/atom command

3.17 compute aggregate/atom command

3.17.1 Syntax

```
compute ID group-ID cluster/atom cutoff
compute ID group-ID fragment/atom keyword value ...
compute ID group-ID aggregate/atom cutoff
```

- ID, group-ID are documented in *compute* command
- *cluster/atom* or *fragment/atom* or *aggregate/atom* = style name of this compute command
- cutoff = distance within which to label atoms as part of same cluster (distance units)
- zero or more keyword/value pairs may be appended to *fragment/atom*
- keyword = *single*

single value = yes or no to treat single atoms (no bonds) as fragments

3.17.2 Examples

```
compute 1 all cluster/atom 3.5
compute 1 all fragment/atom
compute 1 all fragment/atom single no
compute 1 all aggregate/atom 3.5
```

3.17.3 Description

Define a computation that assigns each atom a cluster, fragment, or aggregate ID. Only atoms in the compute group are clustered and assigned cluster IDs. Atoms not in the compute group are assigned an ID = 0.

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

A fragment is similarly defined as a set of atoms, each of which has a bond to another atom in the fragment. Bonds can be defined initially via the *data file* or *create_bonds* commands, or dynamically by fixes which create or break bonds like *fix bond/react*, *fix bond/create*, *fix bond/swap*, or *fix bond/break*. The cluster ID or fragment ID of every atom in the cluster will be set to the smallest atom ID of any atom in the cluster or fragment, respectively.

For the *fragment/atom* style, the *single* keyword determines whether single atoms (not bonded to another atom) are treated as one-atom fragments or not, based on the *yes* or *no* setting. If the setting is *no* (the default), their fragment IDs are set to 0.

An aggregate is defined by combining the rules for clusters and fragments (i.e., a set of atoms, where each of them is within the cutoff distance from one or more atoms within a fragment that is part of the same cluster). This measure can be used to track molecular assemblies like micelles.

For computes *cluster/atom* and *aggregate/atom* a neighbor list needed to compute cluster IDs is constructed each time the compute is invoked. Thus it can be inefficient to compute/dump this quantity too frequently or to have multiple *cluster/atom* or *aggregate/atom* style computes.

Note

If you have a bonded system, then the settings of *special_bonds* command can remove pairwise interactions between atoms in the same bond, angle, or dihedral. This is the default setting for the *special_bonds* command, and means those pairwise interactions do not appear in the neighbor list. Because this fix uses the neighbor list, it also means those pairs will not be included when computing the clusters. This does not apply when using long-range coulomb (*coul/long*, *coul/msm*, *coul/wolf* or similar. One way to get around this would be to set special_bond scaling factors to very tiny numbers that are not exactly zero (e.g., 1.0×10^{-50}). Another workaround is to write a dump file and use the *rerun* command to compute the clusters for snapshots in the dump file. The rerun script can use a *special_bonds* command that includes all pairs in the neighbor list.

Note

For the compute fragment/atom style, each fragment is identified using the current bond topology. This will not account for bonds broken by the *bond_style quartic* command because it does not perform a full update of the bond topology data structures within LAMMPS.

3.17.4 Output info

This compute calculates a per-atom vector, which can be accessed by any command that uses per-atom values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

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

3.17.5 Restrictions

none

3.17.6 Related commands

compute coord/atom

3.17.7 Default

The default for fragment/atom is single=no.

3.18 compute cna/atom command

3.18.1 Syntax

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

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

3.18.2 Examples

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

3.18.3 Description

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

Currently, there are five kinds of CNA patterns LAMMPS recognizes:

- fcc = 1
- hcp = 2
- bcc = 3
- icosahedral = 4

- unknown = 5

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

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

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

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

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

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

Also note that since the CNA calculation in LAMMPS uses the neighbors of an owned atom to find the nearest neighbors of a ghost atom, the following relation should also be satisfied:

$$r_c + r_s > 2 * \mathrm{cutoff}$$

where r_c is the cutoff distance of the potential, r_s is the skin distance as specified by the *neighbor* command, and cutoff is the argument used with the compute cna/atom command. LAMMPS will issue a warning if this is not the case.

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

3.18.4 Output info

This compute calculates a per-atom vector, which can be accessed by any command that uses per-atom values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

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

3.18.5 Restrictions

none

3.18.6 Related commands

compute centro/atom

3.18.7 Default

none

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

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

3.19 compute cnp/atom command

3.19.1 Syntax

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

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

3.19.2 Examples

```
compute 1 all cnp/atom 3.08
```

3.19.3 Description

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

The value of the CNP parameter will be 0.0 for atoms not in the specified compute group. Note that normally a CNP calculation should only be performed on single component systems.

This parameter is computed using the following formula from *(Tsuzuki)*

$$Q_i = \frac{1}{n_i} \sum_{j=1}^{n_i} \left\| \sum_{k=1}^{n_{ij}} \vec{R}_{ik} + \vec{R}_{jk} \right\|^2$$

where the index j goes over the n_i nearest neighbors of atom i, and the index k goes over the n_{ij} common nearest neighbors between atom i and atom j. $\vec{R}_{ik}$ and $\vec{R}_{jk}$ are the vectors connecting atom k to atoms i and j. The quantity in the double sum is computed for each atom.

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

cutoff distance:

$$r_c^{\text{fcc}} = \frac{1}{2}\left(\frac{\sqrt{2}}{2}+1\right)a \approx 0.8536a$$
$$r_c^{\text{bcc}} = \frac{1}{2}(\sqrt{2}+1)a \approx 1.207a$$
$$r_c^{\text{hcp}} = \frac{1}{2}\left(1+\sqrt{\frac{4+2x^2}{3}}\right)a$$

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

Also note that since the CNP calculation in LAMMPS uses the neighbors of an owned atom to find the nearest neighbors of a ghost atom, the following relation should also be satisfied:

$$r_c + r_s > 2 * \text{cutoff}$$

where r_c is the cutoff distance of the potential, r_s is the skin distance as specified by the *neighbor* command, and cutoff is the argument used with the compute cnp/atom command. LAMMPS will issue a warning if this is not the case.

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

3.19.4 Output info

This compute calculates a per-atom vector, which can be accessed by any command that uses per-atom values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

The per-atom vector values will be real positive numbers. Some typical CNP values:

```
FCC lattice = 0.0
BCC lattice = 0.0
HCP lattice = 4.4

FCC (111) surface = 13.0
FCC (100) surface = 26.5
FCC dislocation core = 11
```

3.19.5 Restrictions

This compute is part of the EXTRA-COMPUTE package. It is only enabled if LAMMPS was built with that package. See the *Build package* page for more info.

3.19.6 Related commands

compute cna/atom *compute centro/atom*

3.19.7 Default

none

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

3.20 compute com command

3.20.1 Syntax

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

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

3.20.2 Examples

```
compute 1 all com
```

3.20.3 Description

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

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

Note

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

3.20.4 Output info

This compute calculates a global vector of length 3, which can be accessed by indices 1–3 by any command that uses global vector values from a compute as input. See the *Howto output* doc page for an overview of LAMMPS output options.

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

3.20.5 Restrictions

none

3.20.6 Related commands

compute com/chunk

3.20.7 Default

none

3.21 compute com/chunk command

3.21.1 Syntax

```
compute ID group-ID com/chunk chunkID
```

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

3.21.2 Examples

```
compute 1 fluid com/chunk molchunk
```

3.21.3 Description

Define a computation that calculates the center-of-mass for multiple chunks of atoms.

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* and *Howto chunk* doc pages for details of how chunks can be defined and examples of how they can be used to measure properties of a system.

This compute calculates the (x, y, z) coordinates of the center of mass for each chunk, which includes all effects due to atoms passing through periodic boundaries.

Note that only atoms in the specified group contribute to the calculation. The *compute chunk/atom* command defines its own group; atoms will have a chunk ID = 0 if they are not in that group, signifying they are not assigned to a chunk,

and will thus also not contribute to this calculation. You can specify the "all" group for this command if you simply want to include atoms with non-zero chunk IDs.

Note

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

The simplest way to output the results of the compute com/chunk calculation to a file is to use the *fix ave/time* command, for example:

```
compute cc1 all chunk/atom molecule
compute myChunk all com/chunk cc1
fix 1 all ave/time 100 1 100 c_myChunk[*] file tmp.out mode vector
```

3.21.4 Output info

This compute calculates a global array where the number of rows = the number of chunks *Nchunk* as calculated by the specified *compute chunk/atom* command. The number of columns is 3 for the (x, y, z) center-of-mass coordinates of each chunk. These values can be accessed by any command that uses global array values from a compute as input. See the *Howto output* doc page for an overview of LAMMPS output options.

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

3.21.5 Restrictions

none

3.21.6 Related commands

compute com

3.21.7 Default

none

3.22 compute composition/atom command

Accelerator Variants: *composition/atom/kk*

3.22.1 Syntax

```
compute ID group-ID composition/atom keyword values ...
```

- ID, group-ID are documented in *compute* command
- composition/atom = style name of this compute command
- one or more keyword/value pairs may be appended

  keyword = cutoff
    cutoff value = distance cutoff

3.22.2 Examples

```
compute 1 all composition/atom

compute 1 all composition/atom cutoff 9.0
comm_modify cutoff 9.0
```

3.22.3 Description

Added in version 21Nov2023.

Define a computation that calculates a local composition vector for each atom. For a central atom with M neighbors within the neighbor cutoff sphere, composition is defined as the number of atoms of a given type (including the central atom) divided by (M + 1). For a given central atom, the sum of all compositions equals one.

Note

This compute uses the number of atom types, not chemical species, assigned in *pair_coeff* command. If an interatomic potential has two species (i.e., Cu and Ni) assigned to four different atom types in *pair_coeff* (i.e., 'Cu Cu Ni Ni'), the compute will output four fractional values. In those cases, the user may desire an extra calculation step to consolidate per-type fractions into per-species fractions. This calculation can be conducted within LAMMPS using another compute such as *compute reduce*, an atom-style *variable command*, or as a post-processing step.

The optional keyword *cutoff* defines the distance cutoff used when searching for neighbors. The default value is the cutoff specified by the pair style. If no pair style is defined, then a cutoff must be defined using this keyword. If the specified cutoff is larger than that of the pair_style plus neighbor skin (or no pair style is defined), the *comm_modify cutoff* option must also be set to match that of the *cutoff* keyword.

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

Note

If you have a bonded system, then the settings of *special_bonds* command can remove pairwise interactions between atoms in the same bond, angle, or dihedral. This is the default setting for the *special_bonds* command, and means those pairwise interactions do not appear in the neighbor list. Because this compute uses the neighbor list, it also means those pairs will not be included in the order parameter. This difficulty can be circumvented by writing a

dump file, and using the *rerun* command to compute the order parameter for snapshots in the dump file. The rerun script can use a *special_bonds* command that includes all pairs in the neighbor list.

3.22.4 Output info

This compute calculates a per-atom array with $1 + N$ columns, where N is the number of atom types. The first column is a count of the number of atoms used to calculate composition (including the central atom), and each subsequent column indicates the fraction of that atom type within the cutoff sphere.

These values can be accessed by any command that uses per-atom values from a compute as input. See the *Howto output* doc page for an overview of LAMMPS output options.

3.22.5 Restrictions

This compute is part of the EXTRA-COMPUTE package. It is only enabled if LAMMPS was built with that package. See the *Build package* page for more info.

This compute requires *neighbor styles 'bin' or 'nsq'*.

3.22.6 Related commands

comm_modify

3.22.7 Default

The option defaults are *cutoff* = pair style cutoff.

3.23 compute contact/atom command

3.23.1 Syntax

```
compute ID group-ID contact/atom group2-ID
```

- ID, group-ID are documented in *compute* command
- contact/atom = style name of this compute command
- group2-ID = optional argument to restrict which atoms to consider for contacts (see below)

3.23.2 Examples

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

3.23.3 Description

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

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

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

The optional *group2-ID* argument allows to specify from which group atoms contribute to the coordination number. Default setting is group 'all'.

3.23.4 Output info

This compute calculates a per-atom vector, whose values can be accessed by any command that uses per-atom values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

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

3.23.5 Restrictions

This compute 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 compute requires that atoms store a radius as defined by the *atom_style sphere* command.

3.23.6 Related commands

compute coord/atom

3.23.7 Default

group2-ID = all

3.24 compute coord/atom command

Accelerator Variants: *coord/atom/kk*

3.24.1 Syntax

```
compute ID group-ID coord/atom style args ...
```

- ID, group-ID are documented in *compute* command
- coord/atom = style name of this compute command
- style = *cutoff* or *orientorder*

```
cutoff args = cutoff [group group2-ID] typeN
  cutoff = distance within which to count coordination neighbors (distance units)
  group group2-ID = select group-ID to restrict which atoms to consider for coordination number
  ↪(optional)
  typeN = atom type for Nth coordination count (see asterisk form below)
orientorder args = orientorderID threshold
  orientorderID = ID of an orientorder/atom compute
  threshold = minimum value of the product of two "connected" atoms
```

3.24.2 Examples

```
compute 1 all coord/atom cutoff 2.0
compute 1 all coord/atom cutoff 6.0 1 2
compute 1 all coord/atom cutoff 6.0 2*4 5*8 *
compute 1 solute coord/atom cutoff 2.0 group solvent
compute 1 all coord/atom orientorder 2 0.5
```

3.24.3 Description

This compute performs calculations between neighboring atoms to determine a coordination value. The specific calculation and the meaning of the resulting value depend on the *cstyle* keyword used.

The *cutoff* cstyle calculates one or more traditional coordination numbers for each atom. A coordination number is defined as the number of neighbor atoms with specified atom type(s), and optionally within the specified group, that are within the specified cutoff distance from the central atom. The compute group selects only the central atoms; all neighboring atoms, unless selected by type, type range, or group option, are included in the coordination number tally.

The optional *group* keyword allows to specify from which group atoms contribute to the coordination number. Default setting is group 'all.'

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

The *typeN* keywords can be specified in one of two ways. An explicit numeric value can be used, as in the second example above. Or a wild-card asterisk can be used to specify a range 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 *orientorder* cstyle calculates the number of "connected" neighbor atoms *j* around each central atom *i*. For this *cstyle*, connected is defined by the orientational order parameter calculated by the *compute orientorder/atom* command. This *cstyle* thus allows one to apply the ten Wolde's criterion to identify crystal-like atoms in a system, as discussed in *ten Wolde*.

The ID of the previously specified *compute orientorder/atom* command is specified as *orientorderID*. The compute must invoke its *components* option to calculate components of the *Ybar_lm* vector for each atoms, as described in its documentation. Note that orientorder/atom compute defines its own criteria for identifying neighboring atoms. If the scalar product (*Ybar_lm(i)*, *Ybar_lm(j)*), calculated by the orientorder/atom compute is larger than the specified *threshold*, then *i* and *j* are connected, and the coordination value of *i* is incremented by one.

For all *cstyle* settings, all coordination values will be 0.0 for atoms not in the specified compute group.

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

Note

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

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.

3.24.4 Output info

For *cstyle* cutoff, this compute can calculate a per-atom vector or array. If single *type1* keyword is specified (or if none are specified), this compute calculates a per-atom vector. If multiple *typeN* keywords are specified, this compute calculates a per-atom array, with *N* columns.

For *cstyle* orientorder, this compute calculates a per-atom vector.

These values can be accessed by any command that uses per-atom values from a compute as input. See the *Howto output* doc page for an overview of LAMMPS output options.

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

3.24.5 Restrictions

none

3.24.6 Related commands

compute cluster/atom *compute orientorder/atom*

3.24.7 Default

group = all

(tenWolde) P. R. ten Wolde, M. J. Ruiz-Montero, D. Frenkel, J. Chem. Phys. 104, 9932 (1996).

3.25 compute count/type command

3.25.1 Syntax

```
compute ID group-ID count/type mode
```

- ID, group-ID are documented in *compute* command
- count/type = style name of this compute command
- mode = *atom* or *bond* or *angle* or *dihedral* or *improper*

3.25.2 Examples

```
compute 1 all count/type atom
compute 1 flowmols count/type bond
```

3.25.3 Description

Added in version 15Jun2023.

Define a computation that counts the current number of atoms for each atom type. Or the number of bonds (angles, dihedrals, impropers) for each bond (angle, dihedral, improper) type.

The former can be useful if atoms are added to or deleted from the system in random ways, e.g. via the *fix deposit*, *fix pour*, or *fix evaporate* commands. The latter can be useful in reactive simulations where molecular bonds are broken or created, as well as angles, dihedrals, impropers.

Note that for this command, bonds (angles, etc) are the topological kind enumerated in a data file, initially read by the *read_data* command or defined by the *molecule* command. They do not refer to implicit bonds defined on-the-fly by bond-order or reactive pair styles based on the current conformation of small clusters of atoms.

These commands can turn off topological bonds (angles, etc) by setting their bond (angle, etc) types to negative values. This command includes the turned-off bonds (angles, etc) in the count for each type:

- *fix shake*

- *delete_bonds*

These commands can create and/or break topological bonds (angles, etc). In the case of breaking, they remove the bond (angle, etc) from the system, so that they no longer exist (*bond_style quartic* and *BPM bond styles* are exceptions, see the discussion below). Thus they are not included in the counts for each type:

- *delete_bonds remove*
- *bond_style quartic*
- *fix bond/react*
- *fix bond/create*
- *fix bond/break*
- *BPM package* bond styles

If the *mode* setting is *atom* then the count of atoms for each atom type is tallied. Only atoms in the specified group are counted.

The atom count for each type can be normalized by the total number of atoms like so:

```
compute typevec all count/type atom # number of atoms of each type
variable normtypes vector c_typevec/atoms # divide by total number of atoms
variable ntypes equal extract_setting(ntypes) # number of atom types
thermo_style custom step v_normtypes[*${ntypes}] # vector variable needs upper limit
```

Similarly, bond counts can be normalized by the total number of bonds. The same goes for angles, dihedrals, and impropers (see below).

If the *mode* setting is *bond* then the count of bonds for each bond type is tallied. Only bonds with both atoms in the specified group are counted.

For *mode* = *bond*, broken bonds with a bond type of zero are also counted. The *bond_style quartic* and *BPM bond styles* break bonds by doing this. See the *Howto broken bonds* doc page for more details. Note that the group setting is ignored for broken bonds; all broken bonds in the system are counted.

If the *mode* setting is *angle* then the count of angles for each angle type is tallied. Only angles with all 3 atoms in the specified group are counted.

If the *mode* setting is *dihedral* then the count of dihedrals for each dihedral type is tallied. Only dihedrals with all 4 atoms in the specified group are counted.

If the *mode* setting is *improper* then the count of impropers for each improper type is tallied. Only impropers with all 4 atoms in the specified group are counted.

3.25.4 Output info

This compute calculates a global vector of counts. If the mode is *atom* or *bond* or *angle* or *dihedral* or *improper*, then the vector length is the number of atom types or bond types or angle types or dihedral types or improper types, respectively.

If the mode is *bond* this compute also calculates a global scalar which is the number of broken bonds with type = 0, as explained above.

These values can be used by any command that uses global scalar or vector values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

The scalar and vector values calculated by this compute are both “intensive”.

3.25.5 Restrictions

none

3.25.6 Related commands

none

3.25.7 Default

none

3.26 compute damage/atom command

3.26.1 Syntax

```
compute ID group-ID damage/atom
```

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

3.26.2 Examples

```
compute 1 all damage/atom
```

3.26.3 Description

Define a computation that calculates the per-atom damage for each atom in a group. This is a quantity relevant for *Peridynamics models*. See this document for an overview of LAMMPS commands for Peridynamics modeling.

The "damage" of a Peridynamics particles is based on the bond breakage between the particle and its neighbors. If all the bonds are broken the particle is considered to be fully damaged.

See the *Peridynamics Howto* for a formal definition of "damage" and more details about Peridynamics as it is implemented in LAMMPS.

This command can be used with all the Peridynamic pair styles.

The damage value will be 0.0 for atoms not in the specified compute group.

3.26.4 Output info

This compute calculates a per-atom vector, which can be accessed by any command that uses per-atom values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

The per-atom vector values are unitless numbers (damage) ≥ 0.0.

3.26.5 Restrictions

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

3.26.6 Related commands

compute dilatation/atom, *compute plasticity/atom*

3.26.7 Default

none

3.27 compute dihedral command

3.27.1 Syntax

```
compute ID group-ID dihedral
```

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

3.27.2 Examples

```
compute 1 all dihedral
```

3.27.3 Description

Define a computation that extracts the dihedral energy calculated by each of the dihedral sub-styles used in the *dihedral_style hybrid* command. These values are made accessible for output or further processing by other commands. The group specified for this command is ignored.

This compute is useful when using *dihedral_style hybrid* if you want to know the portion of the total energy contributed by one or more of the hybrid sub-styles.

3.27.4 Output info

This compute calculates a global vector of length *N*, where *N* is the number of sub_styles defined by the *dihedral_style hybrid* command, which can be accessed by the indices 1 through *N*. These values can be used by any command that uses global scalar or vector values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

The vector values are "extensive" and will be in energy *units*.

3.27.5 Restrictions

none

3.27.6 Related commands

compute pe, *compute pair*

3.27.7 Default

none

3.28 compute dihedral/local command

3.28.1 Syntax

```
compute ID group-ID dihedral/local value1 value2 ... keyword args ...
```

- ID, group-ID are documented in *compute* command
- dihedral/local = style name of this compute command
- one or more values may be appended
- value = *phi* or *v_name*

  phi = tabulate dihedral angles
  v_name = equal-style variable with name (see below)

- zero or more keyword/args pairs may be appended
- keyword = *set*

  set args = phi name
    phi = only currently allowed arg
    name = name of variable to set with phi

3.28.2 Examples

```
compute 1 all dihedral/local phi
compute 1 all dihedral/local phi v_cos set phi p
```

3.28.3 Description

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

The value *phi* (ϕ) is the dihedral angle, as defined in the diagram on the *dihedral_style* doc page.

The value *v_name* can be used together with the *set* keyword to compute a user-specified function of the dihedral angle ϕ. The *name* specified for the *v_name* value is the name of an *equal-style variable* which should evaluate a formula based on a variable which will store the angle ϕ. This other variable must be an *internal-style variable* defined in the input script; its initial numeric value can be anything. It must be an internal-style variable, because this command resets its value directly. The *set* keyword is used to identify the name of this other variable associated with ϕ.

Note that the value of ϕ for each angle which stored in the internal variable is in radians, not degrees.

As an example, these commands can be added to the bench/in.rhodo script to compute the $\cos\phi$ and $\cos^2\phi$ of every dihedral angle in the system and output the statistics in various ways:

```
variable p internal 0.0
variable cos equal cos(v_p)
variable cossq equal cos(v_p)*cos(v_p)

compute 1 all property/local datom1 datom2 datom3 datom4 dtype
compute 2 all dihedral/local phi v_cos v_cossq set phi p
dump 1 all local 100 tmp.dump c_1[*] c_2[*]

compute 3 all reduce ave c_2[*]
thermo_style custom step temp press c_3[*]

fix 10 all ave/histo 10 10 100 -1 1 20 c_2[2] mode vector file tmp.histo
```

The *dump local* command will output the angle (ϕ), $\cos(\phi)$, and $\cos^2(\phi)$ for every dihedral in the system. The *thermo_style* command will print the average of those quantities via the *compute reduce* command with thermo output. And the *fix ave/histo* command will histogram the cosine(angle) values and write them to a file.

The local data stored by this command is generated by looping over all the atoms owned on a processor and their dihedrals. A dihedral will only be included if all four atoms in the dihedral are in the specified compute group.

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

Here is an example of how to do this:

```
compute 1 all property/local dtype datom1 datom2 datom3 datom4
compute 2 all dihedral/local phi
dump 1 all local 1000 tmp.dump index c_1[1] c_1[2] c_1[3] c_1[4] c_1[5] c_2[1]
```

3.28.4 Output info

This compute calculates a local vector or local array depending on the number of values. The length of the vector or number of rows in the array is the number of dihedrals. If a single value is specified, a local vector is produced. If two or more values are specified, a local array is produced where the number of columns is equal to the number of values. The vector or array can be accessed by any command that uses local values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

The output for *phi* will be in degrees.

3.28.5 Restrictions

none

3.28.6 Related commands

dump local, *compute property/local*

3.28.7 Default

none

3.29 compute dilatation/atom command

3.29.1 Syntax

```
compute ID group-ID dilatation/atom
```

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

3.29.2 Examples

```
compute 1 all dilatation/atom
```

3.29.3 Description

Define a computation that calculates the per-atom dilatation for each atom in a group. This is a quantity relevant for *Peridynamics models*. See this document for an overview of LAMMPS commands for Peridynamics modeling.

For small deformation, dilatation of is the measure of the volumetric strain.

The dilatation θ for each peridynamic particle i is calculated as a sum over its neighbors with unbroken bonds, where the contribution of the ij pair is a function of the change in bond length (versus the initial length in the reference state), the volume fraction of the particles and an influence function. See the *Peridynamics Howto* for a formal definition of dilatation.

This command can only be used with a subset of the Peridynamic *pair styles*: *peri/lps*, *peri/ves*, and *peri/eps*.

The dilatation value will be 0.0 for atoms not in the specified compute group.

3.29.4 Output info

This compute calculates a per-atom vector, which can be accessed by any command that uses per-atom values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

The per-atom vector values are unitless numbers ($\theta \geq 0.0$).

3.29.5 Restrictions

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

3.29.6 Related commands

compute damage/atom, *compute plasticity/atom*

3.29.7 Default

none

3.30 compute dipole command

3.31 compute dipole/tip4p command

3.31.1 Syntax

```
compute ID group-ID style arg
```

- ID, group-ID are documented in *compute* command
- style = *dipole* or *dipole/tip4p*
- arg = *mass* or *geometry* = use COM or geometric center for charged chunk correction (optional)

3.31.2 Examples

```
compute 1 fluid dipole
compute dw water dipole geometry
compute dw water dipole/tip4p
```

3.31.3 Description

Define a computation that calculates the dipole vector and total dipole for a group of atoms.

These computes calculate the x,y,z coordinates of the dipole vector and the total dipole moment for the atoms in the compute group. This includes all effects due to atoms passing through periodic boundaries. For a group with a net charge the resulting dipole is made position independent by subtracting the position vector of the center of mass or geometric center times the net charge from the computed dipole vector. Both per-atom charges and per-atom dipole moments, if present, contribute to the computed dipole.

Added in version 28Mar2023.

Compute *dipole/tip4p* includes adjustments for the charge carrying point M in molecules with TIP4P water geometry. The corresponding parameters are extracted from the pair style.

Note

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

3.31.4 Output info

These computes calculate a global scalar containing the magnitude of the computed dipole moment and a global vector of length 3 with the dipole vector. See the *Howto output* page for an overview of LAMMPS output options.

The computed values are "intensive". The array values will be in dipole units (i.e., charge *units* times distance *units*).

3.31.5 Restrictions

Compute style *dipole/tip4p* is part of the EXTRA-COMPUTE package. It is only enabled if LAMMPS was built with that package. See the *Build package* page for more info.

Compute style *dipole/tip4p* can only be used with tip4p pair styles.

3.31.6 Related commands

compute dipole/chunk

3.31.7 Default

Using the center of mass is the default setting for the net charge correction.

3.32 compute dipole/chunk command

3.33 compute dipole/tip4p/chunk command

3.33.1 Syntax

```
compute ID group-ID style chunkID arg
```

- ID, group-ID are documented in *compute* command
- style = *dipole/chunk* or *dipole/tip4p/chunk*
- chunkID = ID of *compute chunk/atom* command
- arg = *mass* or *geometry* = use COM or geometric center for charged chunk correction (optional)

3.33.2 Examples

```
compute 1 fluid dipole/chunk molchunk
compute dw water dipole/chunk 1 geometry
```

3.33.3 Description

Define a computation that calculates the dipole vector and total dipole for multiple chunks of atoms.

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* and *Howto chunk* doc pages for details of how chunks can be defined and examples of how they can be used to measure properties of a system.

These computes calculate the (x, y, z) coordinates of the dipole vector and the total dipole moment for each chunk, which includes all effects due to atoms passing through periodic boundaries. For chunks with a net charge the resulting dipole is made position independent by subtracting the position vector of the center of mass or geometric center times the net charge from the computed dipole vector. Both per-atom charges and per-atom dipole moments, if present, contribute to the computed dipole.

Added in version 28Mar2023.

Compute *dipole/tip4p/chunk* includes adjustments for the charge carrying point M in molecules with TIP4P water geometry. The corresponding parameters are extracted from the pair style.

Note that only atoms in the specified group contribute to the calculation. The *compute chunk/atom* command defines its own group; atoms will have a chunk ID = 0 if they are not in that group, signifying they are not assigned to a chunk, and will thus also not contribute to this calculation. You can specify the "all" group for this command if you simply want to include atoms with non-zero chunk IDs.

Note

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

The simplest way to output the results of the compute com/chunk calculation to a file is to use the *fix ave/time* command, for example:

```
compute cc1 all chunk/atom molecule
compute myChunk all dipole/chunk cc1
fix 1 all ave/time 100 1 100 c_myChunk[*] file tmp.out mode vector
```

3.33.4 Output info

These computes calculate a global array where the number of rows = the number of chunks *Nchunk* as calculated by the specified *compute chunk/atom* command. The number of columns is 4 for the (x, y, z) dipole vector components and the total dipole of each chunk. These values can be accessed by any command that uses global array values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

The array values are "intensive". The array values will be in dipole units (i.e., charge *units* times distance *units*).

3.33.5 Restrictions

Compute style *dipole/tip4p/chunk* is part of the EXTRA-COMPUTE package. It is only enabled if LAMMPS was built with that package. See the *Build package* page for more info.

Compute style *dipole/tip4p/chunk* can only be used with tip4p pair styles.

3.33.6 Related commands

compute com/chunk, *compute dipole*

3.33.7 Default

Using the center of mass is the default setting for the net charge correction.

3.34 compute displace/atom command

3.34.1 Syntax

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

- ID, group-ID are documented in *compute* command
- displace/atom = style name of this compute command
- zero or more keyword/arg pairs may be appended
- keyword = *refresh*

  refresh arg = name of per-atom variable

3.34.2 Examples

```
compute 1 all displace/atom
compute 1 all displace/atom refresh myVar
```

3.34.3 Description

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

A vector of four quantities per atom is calculated by this compute. The first three elements of the vector are the (dx, dy, dz) displacements. The fourth component is the total displacement (i.e., $\sqrt{dx^2 + dy^2 + dz^2}$).

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

Note

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

Note

If you want the quantities calculated by this compute to be continuous when running from a *restart file*, then you should use the same ID for this compute, as in the original run. This is so that the fix this compute creates to store per-atom quantities will also have the same ID, and thus be initialized correctly with time = 0 atom coordinates from the restart file.

The *refresh* option can be used in conjunction with the "dump_modify refresh" command to generate incremental dump files.

The definition and motivation of an incremental dump file is as follows. Instead of outputting all atoms at each snapshot (with some associated values), you may only wish to output the subset of atoms with a value that has changed in some way compared to the value the last time that atom was output. In some scenarios this can result in a dramatically smaller dump file. If desired, by post-processing the sequence of snapshots, the values for all atoms at all timesteps can be inferred.

A concrete example using this compute, is a simulation of atom diffusion in a solid, represented as atoms on a lattice. Diffusive hops are rare. Imagine that when a hop occurs an atom moves more than a distance *Dhop*. For any snapshot we only want to output atoms that have hopped since the last snapshot. This can be accomplished with something like the following commands:

```
write_dump      all custom tmp.dump id type x y z    # see comment below

variable        Dhop equal 0.6
variable        check atom "c_dsp[4] > v_Dhop"
compute          dsp all displace/atom refresh check
dump             1 all custom 100 tmp.dump id type x y z
```

(continues on next page)

(continued from previous page)

```
dump_modify     1 append yes thresh c_dsp[4] > ${Dhop} &
                refresh c_dsp delay 100
```

The *dump_modify thresh* command will only output atoms that have displaced more than 0.6 Å on each snapshot (assuming metal units). The dump_modify *refresh* option triggers a call to this compute at the end of every dump.

The *refresh* argument for this compute is the ID of an *atom-style variable* which calculates a Boolean value (0 or 1) based on the same criterion used by dump_modify thresh. This compute evaluates the atom-style variable. For each atom that returns 1 (true), the original (reference) coordinates of the atom (stored by this compute) are updated.

The effect of these commands is that a particular atom will only be output in the dump file on the snapshot after it makes a diffusive hop. It will not be output again until it makes another hop.

Note that in the first snapshot of a subsequent run, no atoms will be typically be output. That is because the initial displacement for all atoms is 0.0. If an initial dump snapshot is desired, containing the initial reference positions of all atoms, one way to do this is illustrated above. An initial write_dump command can be used before the first run. It will contain the positions of all the atoms, Options in the *dump_modify* command above will append new output to that same file and delay the output until a later timestep. The *delay* setting avoids a second time = 0 snapshot which would be empty.

3.34.4 Output info

This compute calculates a per-atom array with four columns, which can be accessed by indices 1–4 by any command that uses per-atom values from a compute as input. See the *Howto output* doc page for an overview of LAMMPS output options.

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

This compute supports the *refresh* option as explained above, for use in conjunction with *dump_modify refresh* to generate incremental dump files.

3.34.5 Restrictions

none

3.34.6 Related commands

compute msd, *dump custom*, *fix store/state*

3.34.7 Default

none

3.35 compute dpd command

3.35.1 Syntax

```
compute ID group-ID dpd
```

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

3.35.2 Examples

```
compute 1 all dpd
```

3.35.3 Description

Define a computation that accumulates the total internal conductive energy (U^{cond}), the total internal mechanical energy (U^{mech}), the total chemical energy (U^{chem}) and the *harmonic* average of the internal temperature (θ_{avg}) for the entire system of particles. See the *compute dpd/atom* command if you want per-particle internal energies and internal temperatures.

The system internal properties are computed according to the following relations:

$$U^{\text{cond}} = \sum_{i=1}^{N} u_i^{\text{cond}}$$

$$U^{\text{mech}} = \sum_{i=1}^{N} u_i^{\text{mech}}$$

$$U^{\text{chem}} = \sum_{i=1}^{N} u_i^{\text{chem}}$$

$$U = \sum_{i=1}^{N} (u_i^{\text{cond}} + u_i^{\text{mech}} + u_i^{\text{chem}})$$

$$\theta_{avg} = \left(\frac{1}{N} \sum_{i=1}^{N} \frac{1}{\theta_i} \right)^{-1}$$

where N is the number of particles in the system.

3.35.4 Output info

This compute calculates a global vector of length 5 (U^{cond}, U^{mech}, U^{chem}, θ_{avg}, N), which can be accessed by indices 1 through 5. See the *Howto output* page for an overview of LAMMPS output options.

The vector values will be in energy and temperature *units*.

3.35.5 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.

3.35.6 Related commands

compute dpd/atom, *thermo_style*

3.35.7 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).

3.36 compute dpd/atom command

3.36.1 Syntax

```
compute ID group-ID dpd/atom
```

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

3.36.2 Examples

```
compute 1 all dpd/atom
```

3.36.3 Description

Define a computation that accesses the per-particle internal conductive energy (u^{cond}), internal mechanical energy (u^{mech}), internal chemical energy (u^{chem}) and internal temperatures (θ) for each particle in a group. See the *compute dpd* command if you want the total internal conductive energy, the total internal mechanical energy, the total chemical energy and average internal temperature of the entire system or group of dpd particles.

3.36.4 Output info

This compute calculates a per-particle array with four columns (u^{cond}, u^{mech}, u^{chem}, θ), which can be accessed by indices 1–4 by any command that uses per-particle values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

The per-particle array values will be in energy (u^{cond}, u^{mech}, u^{chem}) and temperature (θ) *units*.

3.36.5 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.

3.36.6 Related commands

dump custom, *compute dpd*

3.36.7 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).

3.37 compute edpd/temp/atom command

3.37.1 Syntax

```
compute ID group-ID edpd/temp/atom
```

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

3.37.2 Examples

```
compute 1 all edpd/temp/atom
```

3.37.3 Description

Define a computation that calculates the per-atom temperature for each eDPD particle in a group.

The temperature is a local temperature derived from the internal energy of each eDPD particle based on the local equilibrium hypothesis. For more details please see *(Espanol1997)* and *(Li2014)*.

3.37.4 Output info

This compute calculates a per-atom vector, which can be accessed by any command that uses per-atom values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

The per-atom vector values will be in temperature *units*.

3.37.5 Restrictions

This compute is part of the DPD-MESO package. It is only enabled if LAMMPS was built with that package. See the *Build package* page for more info.

3.37.6 Related commands

pair_style edpd

3.37.7 Default

none

(Espanol1997) Espanol, Europhys Lett, 40(6): 631-636 (1997). DOI: 10.1209/epl/i1997-00515-8

(Li2014) Li, Tang, Lei, Caswell, Karniadakis, J Comput Phys, 265: 113-127 (2014). DOI: 10.1016/j.jcp.2014.02.003.

3.38 compute efield/atom command

3.38.1 Syntax

```
compute ID group-ID efield/atom keyword val
```

- ID, group-ID are documented in *compute* command
- efield/atom = style name of this compute command
- zero or more keyword/value pairs may be appended
- keyword = *pair* or *kspace*

  pair args = yes or no
  kspace args = yes or no

3.38.2 Examples

```
compute 1 all efield/atom
compute 1 all efield/atom pair yes kspace no
```

Used in input scripts:

```
examples/PACKAGES/dielectric/in.confined
examples/PACKAGES/dielectric/in.nopbc
```

3.38.3 Description

Define a computation that calculates the electric field at each atom in a group. The compute should only enabled with pair and kspace styles that are provided by the DIELECTRIC package because only these styles compute the per-atom electric field at every time step.

The electric field is a 3-component vector. The value of the electric field components will be 0.0 for atoms not in the specified compute group.

The keyword/value option pairs are used in the following ways.

For the *pair* and *kspace* keywords, the real-space and reciprocal-space contributions to the electric field can be turned off and on.

3.38.4 Output info

This compute calculates a per-atom vector, which can be accessed by any command that uses per-atom values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

The per-atom vector values will be in electric field *units*.

3.38.5 Restrictions

This compute is part of the DIELECTRIC package. It is only enabled if LAMMPS was built with that package.

3.38.6 Related commands

dump custom

3.38.7 Default

The option defaults are pair = yes and kspace = yes.

3.39 compute efield/wolf/atom command

3.39.1 Syntax

```
compute ID group-ID efield/wolf/atom alpha keyword val
```

- ID, group-ID are documented in *compute* command
- efield/atom/wolf = style name of this compute command
- alpha = damping parameter (inverse distance units)
- zero or more keyword/value pairs may be appended
- keyword = *limit* or *cutoff*

  limit group2-ID = limit computing the electric field contributions to a group (default: all)
  cutoff value = set cutoff for computing contributions to this value (default: maximum cutoff of pair_style)

3.39.2 Examples

```
compute 1 all efield/wolf/atom 0.2
compute 1 mols efield/wolf/atom 0.25 limit water cutoff 10.0
```

3.39.3 Description

Added in version 8Feb2023.

Define a computation that approximates the electric field at each atom in a group.

$$\vec{E}_i = \frac{\vec{F}coul_i}{q_i} = \sum_{j \neq i} \frac{q_j}{r_{ij}^2} \qquad r < r_c$$

The electric field at the position of the atom i is the coulomb force on a unit charge at that point, which is equivalent to dividing the Coulomb force by the charge of the individual atom.

In this compute the electric field is approximated as the derivative of the potential energy using the Wolf summation method, described in *Wolf*, given by:

$$E_i = \frac{1}{2} \sum_{j \neq i} \frac{q_i q_j \mathrm{erfc}(\alpha r_{ij})}{r_{ij}} + \frac{1}{2} \sum_{j \neq i} \frac{q_i q_j \mathrm{erf}(\alpha r_{ij})}{r_{ij}} \qquad r < r_c$$

where α is the damping parameter, and *erf()* and *erfc()* are error-function and complementary error-function terms. This potential is essentially a short-range, spherically-truncated, charge-neutralized, force-shifted, pairwise *1/r* summation. With a manipulation of adding and subtracting a self term (for i = j) to the first and second term on the right-hand-side, respectively, and a small enough α damping parameter, the second term shrinks and the potential becomes a rapidly-converging real-space summation. With a long enough cutoff and small enough α parameter, the electric field calculated by the Wolf summation method approaches that computed using the Ewald sum.

The value of the electric field components will be 0.0 for atoms not in the specified compute group.

When the *limit* keyword is used, only contributions from atoms in the selected group will be considered, otherwise contributions from all atoms within the cutoff are included.

When the *cutoff* keyword is used, the cutoff used for the electric field approximation can be set explicitly. By default it is the largest cutoff of any pair style force computation.

Computational Efficiency

This compute will loop over a full neighbor list just like a pair style does when computing forces, thus it can be quite time-consuming and slow down a calculation significantly when its data is used in every time step. The *compute efield/atom* command of the DIELECTRIC package is more efficient in comparison, since the electric field data is collected and stored as part of the force computation at next to no extra computational cost.

3.39.4 Output info

This compute calculates a per-atom vector, which can be accessed by any command that uses per-atom values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

The vector contains 3 values per atom which are the x-, y-, and z-direction electric field components in force units.

3.39.5 Restrictions

This compute is part of the EXTRA-COMPUTE package. It is only enabled if LAMMPS was built with that package.

This compute requires *neighbor styles 'bin' or 'nsq'*.

3.39.6 Related commands

pair_style coul/wolf, *compute efield/atom*

3.39.7 Default

The option defaults are *limit* = all and *cutoff* = largest cutoff for pair styles.

(Wolf) D. Wolf, P. Keblinski, S. R. Phillpot, J. Eggebrecht, J Chem Phys, 110, 8254 (1999).

3.40 compute entropy/atom command

3.40.1 Syntax

```
compute ID group-ID entropy/atom sigma cutoff keyword value ...
```

- ID, group-ID are documented in *compute* command
- entropy/atom = style name of this compute command
- sigma = width of Gaussians used in the $g(r)$ smoothing
- cutoff = cutoff for the $g(r)$ calculation
- one or more keyword/value pairs may be appended

keyword = avg or local
  avg args = neigh cutoff2
    neigh value = yes or no = whether to average the pair entropy over neighbors
    cutoff2 = cutoff for the averaging over neighbors
  local arg = yes or no = use the local density around each atom to normalize the g(r)

3.40.2 Examples

```
compute 1 all entropy/atom 0.25 5.
compute 1 all entropy/atom 0.25 5. avg yes 5.
compute 1 all entropy/atom 0.125 7.3 avg yes 5.1 local yes
```

3.40.3 Description

Define a computation that calculates the pair entropy fingerprint for each atom in the group. The fingerprint is useful to distinguish between ordered and disordered environments, for instance liquid and solid-like environments, or glassy and crystalline-like environments. Some applications could be the identification of grain boundaries, a melt-solid interface, or a solid cluster emerging from the melt. The advantage of this parameter over others is that no a priori information about the solid structure is required.

This parameter for atom i is computed using the following formula from *(Piaggi)* and *(Nettleton)* ,

$$s_S^i = -2\pi\rho k_B \int_0^{r_m} \left[g(r)\ln g(r) - g(r) + 1\right] r^2 dr$$

where r is a distance, $g(r)$ is the radial distribution function of atom i, and ρ is the density of the system. The $g(r)$ computed for each atom i can be noisy and therefore it is smoothed using

$$g_m^i(r) = \frac{1}{4\pi\rho r^2} \sum_j \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(r-r_{ij})^2/(2\sigma^2)}$$

where the sum over j goes through the neighbors of atom i and σ is a parameter to control the smoothing.

The input parameters are *sigma* the smoothing parameter σ, and the *cutoff* for the calculation of $g(r)$.

If the keyword *avg* has the setting *yes*, then this compute also averages the parameter over the neighbors of atom i according to

$$\left\langle s_S^i \right\rangle = \frac{\sum_j s_S^j + s_S^i}{N+1},$$

where the sum over j goes over the neighbors of atom i and N is the number of neighbors. This procedure provides a sharper distinction between order and disorder environments. In this case the input parameter *cutoff2* is the cutoff for the averaging over the neighbors and must also be specified.

If the *avg yes* option is used, the effective cutoff of the neighbor list should be *cutoff*+*cutoff2* and therefore it might be necessary to increase the skin of the neighbor list with:

```
neighbor <skin distance> bin
```

See *neighbor* for details.

If the *local yes* option is used, the $g(r)$ is normalized by the local density around each atom, that is to say the density around each atom is the number of neighbors within the neighbor list cutoff divided by the corresponding volume. This option can be useful when dealing with inhomogeneous systems such as those that have surfaces.

Here are typical input parameters for fcc aluminum (lattice constant 4.05 Å),

```
compute 1 all entropy/atom 0.25 5.7 avg yes 3.7
```

and for bcc sodium (lattice constant 4.23 Å),

```
compute 1 all entropy/atom 0.25 7.3 avg yes 5.1
```

3.40.4 Output info

By default, this compute calculates the pair entropy value for each atom as a per-atom vector, which can be accessed by any command that uses per-atom values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

The pair entropy values have units of the Boltzmann constant. They are always negative, and lower values (lower entropy) correspond to more ordered environments.

3.40.5 Restrictions

This compute is part of the EXTRA-COMPUTE package. It is only enabled if LAMMPS was built with that package. See the *Build package* page for more info.

3.40.6 Related commands

compute cna/atom compute centro/atom

3.40.7 Default

The default values for the optional keywords are avg = no and local = no.

(Piaggi) Piaggi and Parrinello, J Chem Phys, 147, 114112 (2017).

(Nettleton) Nettleton and Green, J Chem Phys, 29, 6 (1958).

3.41 compute erotate/asphere command

3.41.1 Syntax

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

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

3.41.2 Examples

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

3.41.3 Description

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

For all 3 types of particles, the rotational kinetic energy is computed as $\frac{1}{2} I \omega^2$, where I is the inertia tensor for the aspherical particle and ω is its angular velocity, which is computed from its angular momentum if needed.

Note

For *2d models*, ellipsoidal particles are treated as ellipsoids, not ellipses, meaning their moments of inertia will be the same as in 3d.

3.41.4 Output info

This compute calculates a global scalar (the KE). This value can be used by any command that uses a global scalar value from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

The scalar value calculated by this compute is "extensive". The scalar value will be in energy *units*.

3.41.5 Restrictions

This compute requires that ellipsoidal particles atoms store a shape and quaternion orientation and angular momentum as defined by the *atom_style ellipsoid* command.

This compute requires that line segment particles atoms store a length and orientation and angular velocity as defined by the *atom_style line* command.

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

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

3.41.6 Related commands

none

compute erotate/sphere

3.41.7 Default

none

3.42 compute erotate/rigid command

3.42.1 Syntax

```
compute ID group-ID erotate/rigid fix-ID
```

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

3.42.2 Examples

```
compute 1 all erotate/rigid myRigid
```

3.42.3 Description

Define a computation that calculates the rotational kinetic energy of a collection of rigid bodies, as defined by one of the *fix rigid* command variants.

The rotational energy of each rigid body is computed as $\frac{1}{2} I \omega_{\text{body}}^2$, where I is the inertia tensor for the rigid body and ω_{body} is its angular velocity vector. Both I and ω_{body} are in the frame of reference of the rigid body (i.e., I is diagonal).

The *fix-ID* should be the ID of one of the *fix rigid* commands which defines the rigid bodies. The group specified in the compute command is ignored. The rotational energy of all the rigid bodies defined by the fix rigid command in included in the calculation.

3.42.4 Output info

This compute calculates a global scalar (the summed rotational energy of all the rigid bodies). This value can be used by any command that uses a global scalar value from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

The scalar value calculated by this compute is "extensive". The scalar value will be in energy *units*.

3.42.5 Restrictions

This compute 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.

3.42.6 Related commands

compute ke/rigid

3.42.7 Default

none

3.43 compute erotate/sphere command

Accelerator Variants: *erotate/sphere/kk*

3.43.1 Syntax

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

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

3.43.2 Examples

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

3.43.3 Description

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

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

Note

For *2d models*, particles are treated as spheres, not disks, meaning their moment of inertia will be the same as in 3d.

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.

3.43.4 Output info

This compute calculates a global scalar (the KE). This value can be used by any command that uses a global scalar value from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

The scalar value calculated by this compute is "extensive". The scalar value will be in energy *units*.

3.43.5 Restrictions

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

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

3.43.6 Related commands

compute erotate/asphere

3.43.7 Default

none

3.44 compute erotate/sphere/atom command

3.44.1 Syntax

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

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

3.44.2 Examples

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

3.44.3 Description

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

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

> **Note**
>
> For *2d models*, particles are treated as spheres, not disks, meaning their moment of inertia will be the same as in 3d.

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

3.44.4 Output info

This compute calculates a per-atom vector, which can be accessed by any command that uses per-atom values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

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

3.44.5 Restrictions

none

3.44.6 Related commands

dump custom

3.44.7 Default

none

3.45 compute event/displace command

3.45.1 Syntax

```
compute ID group-ID event/displace threshold
```

- ID, group-ID are documented in *compute* command
- event/displace = style name of this compute command
- threshold = minimum distance any particle must move to trigger an event (distance units)

3.45.2 Examples

```
compute 1 all event/displace 0.5
```

3.45.3 Description

Define a computation that flags an "event" if any particle in the group has moved a distance greater than the specified threshold distance when compared to a previously stored reference state (i.e., the previous event). This compute is typically used in conjunction with the *prd* and *tad* commands, to detect if a transition to a new minimum energy basin has occurred.

This value calculated by the compute is equal to 0 if no particle has moved far enough, and equal to 1 if one or more particles have moved further than the threshold distance.

Note

If the system is undergoing significant center-of-mass motion, due to thermal motion, an external force, or an initial net momentum, then this compute will not be able to distinguish that motion from local atom displacements and may generate "false positives".

3.45.4 Output info

This compute calculates a global scalar (the flag). This value can be used by any command that uses a global scalar value from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

The scalar value calculated by this compute is "intensive". The scalar value will be a 0 or 1 as explained above.

3.45.5 Restrictions

This command can only be used if LAMMPS was built with the REPLICA package. See the *Build package* doc page for more info.

3.45.6 Related commands

prd, *tad*

3.45.7 Default

none

3.46 compute fabric command

3.46.1 Syntax

```
compute ID group-ID fabric cutoff attribute ... keyword values ...
```

- ID, group-ID are documented in *compute* command
- fabric = style name of this compute command
- cutoff = *type* or *radius*

  type = cutoffs determined based on atom types
  radius = cutoffs determined based on atom diameters (atom style sphere)

- one or more attributes may be appended
- attribute = *contact* or *branch* or *force/normal* or *force/tangential*

  contact = contact tensor
  branch = branch tensor
  force/normal = normal force tensor
  force/tangential = tangential force tensor

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

  type/include value = arg1 arg2
    arg = separate lists of types (see below)

3.46.2 Examples

```
compute 1 all fabric type contact force/normal type/include 1,2 3*4
compute 1 all fabric radius force/normal force/tangential
```

3.46.3 Description

Define a compute that calculates various fabric tensors for pairwise interaction *(Ouadfel)*. Fabric tensors are commonly used to quantify the anisotropy or orientation of granular contacts but can also be used to characterize the direction of pairwise interactions in general systems. The *type* and *radius* settings are used to select whether interactions cutoffs are determined by atom types or by the sum of atomic radii (atom style sphere), respectively. Calling this compute is roughly the cost of a pair style invocation as it involves a loop over the neighbor list. If the normal or tangential force tensors are requested, it will be more expensive than a pair style invocation as it will also recalculate all pair forces.

Four fabric tensors are available: the contact, branch, normal force, or tangential force tensor. The contact tensor is calculated as

$$C_{ab} = \frac{15}{2}(\phi_{ab} - \frac{1}{3}\mathrm{Tr}(\phi)\delta_{ab})$$

where a and b are the x, y, z directions, δ_{ab} is the Kronecker delta function, and the tensor ϕ is defined as

$$\phi_{ab} = \sum_{n=1}^{N_p} \frac{r_a r_b}{r^2}$$

where n loops over the N_p pair interactions in the simulation, r_a is the a component of the radial vector between the two pairwise interacting particles, and r is the magnitude of the radial vector.

The branch tensor is calculated as

$$B_{ab} = \frac{15}{2\,\mathrm{Tr}(D)}(D_{ab} - \frac{1}{3}\mathrm{Tr}(D)\delta_{ab})$$

where the tensor D is defined as

$$D_{ab} = \sum_{n=1}^{N_p} \frac{1}{N_c(r^2 + C_{cd}r_c r_d)} \frac{r_a r_b}{r}$$

where N_c is the total number of contacts in the system and the subscripts c and d indices are summed according to Einstein notation.

The normal force fabric tensor is calculated as

$$F^n_{ab} = \frac{15}{2\,\mathrm{Tr}(N)}(N_{ab} - \frac{1}{3}\mathrm{Tr}(N)\delta_{ab})$$

where the tensor N is defined as

$$N_{ab} = \sum_{n=1}^{N_p} \frac{1}{N_c(r^2 + C_{cd}r_c r_d)} \frac{r_a r_b}{r^2} f_n$$

and f_n is the magnitude of the normal, central-body force between the two atoms.

Finally, the tangential force fabric tensor is only defined for pair styles that apply tangential forces to particles, namely granular pair styles. It is calculated as

$$F^t_{ab} = \frac{5}{\mathrm{Tr}(N)}(T_{ab} - \frac{1}{3}\mathrm{Tr}(T)\delta_{ab})$$

where the tensor T is defined as

$$T_{ab} = \sum_{n=1}^{N_p} \frac{1}{N_c(r^2 + C_{cd}r_c r_d)} \frac{r_a r_b}{r^2} f_t$$

and f_t is the magnitude of the tangential force between the two atoms.

The *type/include* keyword filters interactions based on the types of the two atoms. Interactions between two atoms are only included in calculations if the atom types are in the two lists. Each list consists of a series of type ranges separated by commas. The range can be specified as a single numeric value, or a wildcard asterisk can be used to specify a range of values. This takes the form "*" or "*n" or "m*" or "m*n". For example, if M is the number of atom types, then an asterisk with no numeric values means all types from 1 to M. A leading asterisk means all types from 1 to n (inclusive). A trailing asterisk means all types from m to M (inclusive). A middle asterisk means all types from m to n (inclusive). Multiple *type/include* keywords may be added.

3.46.4 Output info

This compute calculates a global vector of doubles and a global scalar. The vector stores the unique components of the first requested tensor in the order *xx*, *yy*, *zz*, *xy*, *xz*, *yz* followed by the same components for all subsequent tensors. The length of the vector is therefore six times the number of requested tensors. The scalar output is the number of pairwise interactions included in the calculation of the fabric tensor.

3.46.5 Restrictions

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

Currently, compute *fabric* does not support pair styles with many-body interactions. It also does not support models with long-range Coulombic or dispersion forces, i.e. the kspace_style command in LAMMPS. It also does not support the following fixes which add rigid-body constraints: *fix shake*, *fix rattle*, *fix rigid*, *fix rigid/small*. It does not support granular pair styles that extend beyond the contact of atomic radii (e.g., JKR and DMT).

3.46.6 Related commands

none

3.46.7 Default

none

(Ouadfel) Ouadfel and Rothenburg "Stress-force-fabric relationship for assemblies of ellipsoids", Mechanics of Materials (2001). (link to paper)

3.47 compute fep command

3.47.1 Syntax

```
compute ID group-ID fep temp attribute args ... keyword value ...
```

- ID, group-ID are documented in the *compute* command
- fep = name of this compute command
- temp = external temperature (as specified for constant-temperature run)
- one or more attributes with args may be appended
- attribute = *pair* or *atom*

  pair args = pstyle pparam I J v_delta
    pstyle = pair style name (e.g., lj/cut)
    pparam = parameter to perturb
    I,J = type pair(s) to set parameter for
    v_delta = variable with perturbation to apply (in the units of the parameter)
  atom args = aparam I v_delta
    aparam = charge = parameter to perturb
    I = type to set parameter for
    v_delta = variable with perturbation to apply (in the units of the parameter)
- zero or more keyword/value pairs may be appended
- keyword = *tail* or *volume*

tail value = no or yes
  no = ignore tail correction to pair energies (usually small in fep)
  yes = include tail correction to pair energies
volume value = no or yes
  no = ignore volume changes (e.g., in NVE or NVT trajectories)
  yes = include volume changes (e.g., in NPT trajectories)

3.47.2 Examples

```
compute 1 all fep 298 pair lj/cut epsilon 1 * v_delta pair lj/cut sigma 1 * v_delta volume yes
compute 1 all fep 300 atom charge 2 v_delta
```

Example input scripts available: examples/PACKAGES/fep

3.47.3 Description

Apply a perturbation to parameters of the interaction potential and recalculate the pair potential energy without changing the atomic coordinates from those of the reference, unperturbed system. This compute can be used to calculate free energy differences using several methods, such as free-energy perturbation (FEP), finite-difference thermodynamic integration (FDTI) or Bennet's acceptance ratio method (BAR).

The potential energy of the system is decomposed in three terms: a background term corresponding to interaction sites whose parameters remain constant, a reference term U_0 corresponding to the initial interactions of the atoms that will undergo perturbation, and a term U_1 corresponding to the final interactions of these atoms:

$$U(\lambda) = U_{\mathrm{bg}} + U_1(\lambda) + U_0(\lambda)$$

A coupling parameter λ varying from 0 to 1 connects the reference and perturbed systems:

$$\begin{aligned} \lambda = 0 \quad &\Rightarrow \quad U = U_{\mathrm{bg}} + U_0 \\ \lambda = 1 \quad &\Rightarrow \quad U = U_{\mathrm{bg}} + U_1 \end{aligned}$$

It is possible but not necessary that the coupling parameter (or a function thereof) appears as a multiplication factor of the potential energy. Therefore, this compute can apply perturbations to interaction parameters that are not directly proportional to the potential energy (e.g., σ in Lennard-Jones potentials).

This command can be combined with *fix adapt* to perform multistage free-energy perturbation calculations along stepwise alchemical transformations during a simulation run:

$$\Delta_0^1 A = \sum_{i=0}^{n-1} \Delta_{\lambda_i}^{\lambda_{i+1}} A = -k_B T \sum_{i=0}^{n-1} \ln \left\langle \exp\left(-\frac{U(\lambda_{i+1}) - U(\lambda_i)}{k_B T} \right) \right\rangle_{\lambda_i}$$

This compute is suitable for the finite-difference thermodynamic integration (FDTI) method *(Mezei)*, which is based on an evaluation of the numerical derivative of the free energy by a perturbation method using a very small δ:

$$\Delta_0^1 A = \int_{\lambda=0}^{\lambda=1} \left(\frac{\partial A(\lambda)}{\partial \lambda} \right)_\lambda \mathrm{d}\lambda \approx \sum_{i=0}^{n-1} w_i \frac{A(\lambda_i + \delta) - A(\lambda_i)}{\delta}$$

where w_i are weights of a numerical quadrature. The *fix adapt* command can be used to define the stages of λ at which the derivative is calculated and averaged.

The compute fep calculates the exponential Boltzmann term and also the potential energy difference $U_1 - U_0$. By choosing a very small perturbation δ the thermodynamic integration method can be implemented using a numerical evaluation of the derivative of the potential energy with respect to λ:

$$\Delta_0^1 A = \int_{\lambda=0}^{\lambda=1} \left\langle \frac{\partial U(\lambda)}{\partial \lambda} \right\rangle_\lambda \mathrm{d}\lambda \approx \sum_{i=0}^{n-1} w_i \left\langle \frac{U(\lambda_i + \delta) - U(\lambda_i)}{\delta} \right\rangle_{\lambda_i}$$

Another technique to calculate free energy differences is the acceptance ratio method *(Bennet)*, which can be implemented by calculating the potential energy differences with $\delta = 1.0$ on both the forward and reverse routes:

$$\left\langle \frac{1}{1+\exp\left[\left(U_1 - U_0 - \Delta_0^1 A\right)/k_B T\right]} \right\rangle_0 = \left\langle \frac{1}{1+\exp\left[\left(U_0 - U_1 + \Delta_0^1 A\right)/k_B T\right]} \right\rangle_1$$

The value of the free energy difference is determined by numerical root finding to establish the equality.

Concerning the choice of how the atomic parameters are perturbed in order to setup an alchemical transformation route, several strategies are available, such as single-topology or double-topology strategies *(Pearlman)*. The latter does not require modification of bond lengths, angles or other internal coordinates.

NOTES: This compute command does not take kinetic energy into account, therefore the masses of the particles should not be modified between the reference and perturbed states, or along the alchemical transformation route. This compute command does not change bond lengths or other internal coordinates *(Boresch, Karplus)*.

The *pair* attribute enables various parameters of potentials defined by the *pair_style* and *pair_coeff* commands to be changed, if the pair style supports it.

The *pstyle* argument is the name of the pair style. For example, *pstyle* could be specified as "lj/cut". The *pparam* argument is the name of the parameter to change. This is a list of pair styles and parameters that can be used with this compute. See the doc pages for individual pair styles and their energy formulas for the meaning of these parameters:

born	a,b,c	type pairs
buck, buck/coul/cut, buck/coul/long, buck/coul/msm	a,c	type pairs
buck/mdf	a,c	type pairs
coul/cut	scale	type pairs
coul/cut/soft	lambda	type pairs
coul/long, coul/msm	scale	type pairs
coul/long/soft	scale, lambda	type pairs
eam	scale	type pairs
gauss	a	type pairs
lennard/mdf	a,b	type pairs
lj/class2	epsilon,sigma	type pairs
lj/class2/coul/cut, lj/class2/coul/long	epsilon,sigma	type pairs
lj/cut	epsilon,sigma	type pairs
lj/cut/soft	epsilon,sigma,lambda	type pairs
lj/cut/coul/cut, lj/cut/coul/long, lj/cut/coul/msm	epsilon,sigma	type pairs
lj/cut/coul/cut/soft, lj/cut/coul/long/soft	epsilon,sigma,lambda	type pairs
lj/cut/tip4p/cut, lj/cut/tip4p/long	epsilon,sigma	type pairs
lj/cut/tip4p/long/soft	epsilon,sigma,lambda	type pairs
lj/expand	epsilon,sigma,delta	type pairs
lj/mdf	epsilon,sigma	type pairs
lj/sf/dipole/sf	epsilon,sigma,scale	type pairs
mie/cut	epsilon,sigma,gamR,gamA	type pairs
morse, morse/smooth/linear	d0,r0,alpha	type pairs
morse/soft	d0,r0,alpha,lambda	type pairs
nm/cut	e0,r0,nn,mm	type pairs
nm/cut/coul/cut, nm/cut/coul/long	e0,r0,nn,mm	type pairs
ufm	epsilon,sigma,scale	type pairs
soft	a	type pairs

Note that 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.

Similar to the *pair_coeff* command, I and J can be specified in one of two ways. Explicit numeric values can be used for each, as in the first example above. I $\leq$ J is required. LAMMPS sets the coefficients for the symmetric J,I interaction to the same values. A wild-card asterisk can be used in place of or in conjunction with the I,J arguments to set the coefficients for multiple pairs of atom types. This takes the form "*" or "*n" or "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). Note that only type pairs with I $\leq$ J are considered; if asterisks imply type pairs where J $<$ I, they are ignored.

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 compute is invoked. It should be specified as v_name, where name is the variable name.

The *atom* attribute enables 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 used with this compute:

- charge = charge on particle

The *v_name* argument for keyword *pair* is the name of an *equal-style variable* which will be evaluated each time this compute is invoked. It should be specified as v_name, where name is the variable name.

The *tail* keyword controls the calculation of the tail correction to "van der Waals" pair energies beyond the cutoff, if this has been activated via the *pair_modify* command. If the perturbation is small, the tail contribution to the energy difference between the reference and perturbed systems should be negligible.

If the keyword *volume* = *yes*, then the Boltzmann term is multiplied by the volume so that correct ensemble averaging can be performed over trajectories during which the volume fluctuates or changes *(Allen and Tildesley)*:

$$\Delta_0^1 A = -k_B T \sum_{i=0}^{n-1} \ln \frac{\left\langle V \exp\left(-\frac{U(\lambda_{i+1})-U(\lambda_i)}{k_B T}\right)\right\rangle_{\lambda_i}}{\langle V \rangle_{\lambda_i}}$$

3.47.4 Output info

This compute calculates a global vector of length 3 which contains the energy difference ($U_1 - U_0$) as c_ID[1], the Boltzmann factor $\exp(-(U_1 - U_0)/k_B T)$, or $V \exp(-(U_1 - U_0)/k_B T)$, as c_ID[2] and the volume of the simulation box V as c_ID[3]. U_1 is the pair potential energy obtained with the perturbed parameters and U_0 is the pair potential energy obtained with the unperturbed parameters. The energies include kspace terms if these are used in the simulation.

These output results can be used by any command that uses a global scalar or vector from a compute as input. See the *Howto output* page for an overview of LAMMPS output options. For example, the computed values can be averaged using *fix ave/time*.

The values calculated by this compute are "extensive".

3.47.5 Restrictions

This compute is distributed as the FEP package. It is only enabled if LAMMPS was built with that package. See the *Build package* page for more info.

3.47.6 Related commands

fix adapt/fep, *fix ave/time*, *pair_style . . . /soft*

3.47.7 Default

The option defaults are *tail* = *no*, *volume* = *no*.

(Pearlman) Pearlman, J Chem Phys, 98, 1487 (1994)

(Mezei) Mezei, J Chem Phys, 86, 7084 (1987)

(Bennet) Bennet, J Comput Phys, 22, 245 (1976)

(BoreschKarplus) Boresch and Karplus, J Phys Chem A, 103, 103 (1999)

(AllenTildesley) Allen and Tildesley, Computer Simulation of Liquids, Oxford University Press (1987)

3.48 compute fep/ta command

3.48.1 Syntax

```
compute ID group-ID fep/ta temp plane scale_factor keyword value ...
```

- ID, group-ID are documented in the *compute* command
- fep/ta = name of this compute command
- temp = external temperature (as specified for constant-temperature run)
- plane = *xy* or *xz* or *yz*
- scale_factor = multiplicative factor for change in plane area
- zero or more keyword/value pairs may be appended
- keyword = *tail*

  tail value = no or yes
    no = ignore tail correction to pair energies (usually small in fep)
    yes = include tail correction to pair energies

3.48.2 Examples

```
compute 1 all fep/ta 298 xy 1.0005
```

3.48.3 Description

Added in version 4May2022.

Define a computation that calculates the change in the free energy due to a test-area (TA) perturbation *(Gloor)*. The test-area approach can be used to determine the interfacial tension of the system in a single simulation:

$$\gamma = \lim_{\Delta\mathcal{A}\to 0}\left(\frac{\Delta A_{0\to 1}}{\Delta\mathcal{A}}\right)_{N,V,T} = -\frac{k_B T}{\Delta\mathcal{A}}\ln\left\langle \exp\left(\frac{-(U_1-U_0)}{k_B T}\right)\right\rangle_0$$

During the perturbation, both axes of *plane* are scaled by multiplying $\sqrt{\text{scale\_factor}}$, while the other axis divided by scale_factor such that the overall volume of the system is maintained.

The *tail* keyword controls the calculation of the tail correction to "van der Waals" pair energies beyond the cutoff, if this has been activated via the *pair_modify* command. If the perturbation is small, the tail contribution to the energy difference between the reference and perturbed systems should be negligible.

3.48.4 Output info

This compute calculates a global vector of length 3 which contains the energy difference $(U_1 - U_0)$ as c_ID[1], the Boltzmann factor $\exp\left(-(U_1-U_0)/k_B T\right)$, as c_ID[2] and the change in the *plane* area $\Delta\mathcal{A}$ as c_ID[3]. U_1 is the potential energy of the perturbed state and U_0 is the potential energy of the reference state. The energies include kspace terms if these are used in the simulation.

These output results can be used by any command that uses a global scalar or vector from a compute as input. See the *Howto output* page for an overview of LAMMPS output options. For example, the computed values can be averaged using *fix ave/time*.

3.48.5 Restrictions

Constraints, like fix shake, may lead to incorrect values for energy difference.

This compute is distributed as the FEP package. It is only enabled if LAMMPS was built with that package. See the *Build package* page for more info.

3.48.6 Related commands

compute fep

3.48.7 Default

The option defaults are *tail* = *no*.

(Gloor) Gloor, J Chem Phys, 123, 134703 (2005)

3.49 compute global/atom command

3.49.1 Syntax

```
compute ID group-ID style index input1 input2 ...
```

- ID, group-ID are documented in *compute* command
- global/atom = style name of this compute command
- index = c_ID, c_ID[N], f_ID, f_ID[N], v_name

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

- one or more inputs can be listed
- input = c_ID, c_ID[N], f_ID, f_ID[N], v_name

```
c_ID = global vector calculated by a compute with ID
c_ID[I] = Ith column of global array calculated by a compute with ID, I can include wildcard (see␣
↪below)
f_ID = global vector calculated by a fix with ID
f_ID[I] = Ith column of global array calculated by a fix with ID, I can include wildcard (see below)
v_name = global vector calculated by a vector-style variable with name
```

3.49.2 Examples

```
compute 1 all global/atom c_chunk c_com[1] c_com[2] c_com[3]
compute 1 all global/atom c_chunk c_com[*]
```

3.49.3 Description

Define a calculation that assigns global values to each atom from vectors or arrays of global values. The specified *index* parameter is used to determine which global value is assigned to each atom.

The *index* parameter must reference a per-atom vector or array from a *compute* or *fix* or the evaluation of an atom-style *variable*. Each *input* value must reference a global vector or array from a *compute* or *fix* or the evaluation of an vector-style *variable*. Details are given below.

The *index* value for an atom is used as an index I (from 1 to N, where N is the number of atoms) into the vector associated with each of the input values. The Ith value from the input vector becomes one output value for that atom.

If the atom is not in the specified group, or the index $I < 1$ or $I > M$, where M is the actual length of the input vector, then an output value of 0.0 is assigned to the atom.

An example of how this command is useful, is in the context of "chunks" which are static or dynamic subsets of atoms. The *compute chunk/atom* command assigns unique chunk IDs to each atom. Its output can be used as the *index* parameter for this command. Various other computes with "chunk" in their style name, such as *compute com/chunk* or *compute msd/chunk*, calculate properties for each chunk. The output of these commands are global vectors or arrays, with one or more values per chunk, and can be used as input values for this command. This command will then assign the global chunk value to each atom in the chunk, producing a per-atom vector or per-atom array as output. The per-atom values can then be output to a dump file or used by any command that uses per-atom values from a compute as input, as discussed on the *Howto output* doc page.

As a concrete example, these commands will calculate the displacement of each atom from the center-of-mass of the molecule it is in, and dump those values to a dump file. In this case, each molecule is a chunk.

```
compute cc1 all chunk/atom molecule
compute myChunk all com/chunk cc1
compute prop all property/atom xu yu zu
compute glob all global/atom c_cc1 c_myChunk[*]
variable dx atom c_prop[1]-c_glob[1]
variable dy atom c_prop[2]-c_glob[2]
variable dz atom c_prop[3]-c_glob[3]
variable dist atom sqrt(v_dx*v_dx+v_dy*v_dy+v_dz*v_dz)
dump 1 all custom 100 tmp.dump id xu yu zu c_glob[1] c_glob[2] c_glob[3] &
    v_dx v_dy v_dz v_dist
dump_modify 1 sort id
```

You can add these commands to the bench/in.chain script to see how they work.

Note that for input 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, the following two compute global/atom commands are equivalent, since the *compute com/chunk* command creates a global array with three columns:

```
compute cc1 all chunk/atom molecule
compute com all com/chunk cc1
compute 1 all global/atom c_cc1 c_com[1] c_com[2] c_com[3]
compute 1 all global/atom c_cc1 c_com[*]
```

This section explains the *index* parameter. Note that it must reference per-atom values, as contrasted with the *input* values, which must reference global values.

Note that all of these options generate floating point values. When they are used as an index into the specified input vectors, they simple rounded down to convert the value to integer indices. The final values should range from 1 to N (inclusive), since they are used to access values from N-length vectors.

If *index* begins with "c_", a compute ID must follow which has been previously defined in the input script. The compute must generate per-atom quantities. See the individual *compute* doc page for details. 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 *index* begins with "f_", a fix ID must follow which has been previously defined in the input script. The Fix must generate per-atom quantities. See the individual *fix* page for details. Note that some fixes only produce their values on certain timesteps, which must be compatible with when compute global/atom references the values, else an error results. 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. Users can also write code for their own fix style 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 *index* begins with "v_", a variable name must follow which has been previously defined in the input script. It must be an *atom-style variable*. Atom-style variables can reference thermodynamic keywords and various per-atom attributes, or invoke other computes, fixes, or variables when they are evaluated, so this is a very general means of generating per-atom quantities to use as *index*.

This section explains the kinds of *input* values that can be used. Note that inputs reference global values, as contrasted with the *index* parameter which must reference per-atom values.

If a value begins with "c_", a compute ID must follow which has been previously defined in the input script. The compute must generate a global vector or array. See the individual *compute* doc page for details. If no bracketed integer is appended, the vector calculated by the compute is used. If a bracketed integer is appended, the Ith column of the array calculated by the compute is used. Users can also write code for their own compute styles and *add them to 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. The fix must generate a global vector or array. See the individual *fix* doc page for details. Note that some fixes only produce their values on certain timesteps, which must be compatible with when compute global/atom references the values, else an error results. If no bracketed integer is appended, the vector calculated by the fix is used. If a bracketed integer is appended, the Ith column of the array calculated by the fix is used. Users can also write code for their own fix style and *add them to LAMMPS*. 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. It must be a *vector-style variable*. Vector-style variables can reference thermodynamic keywords and various other attributes of atoms, or invoke other computes, fixes, or variables when they are evaluated, so this is a very general means of generating a vector of global quantities which the *index* parameter will reference for assignment of global values to atoms.

3.49.4 Output info

If a single input is specified this compute produces a per-atom vector. If multiple inputs are specified, this compute produces a per-atom array values, where the number of columns is equal to the number of inputs specified. These values can be used by any command that uses per-atom vector or array values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

The per-atom vector or array values will be in whatever units the corresponding input values are in.

3.49.5 Restrictions

none

3.49.6 Related commands

compute, *fix*, *variable*, *compute chunk/atom*, *compute reduce*

3.49.7 Default

none

3.50 compute group/group command

3.50.1 Syntax

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

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

  pair value = yes or no
  kspace value = yes or no
  boundary value = yes or no
  molecule value = off or inter or intra

3.50.2 Examples

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

3.50.3 Description

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

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

Note

The energies computed by the *pair* keyword do not include tail corrections, even if they are enabled via the *pair_modify* command.

If the *molecule* keyword is set to *inter* or *intra* than an additional check is made based on the molecule IDs of the two atoms in each pair before including their pairwise interaction energy and force. For the *inter* setting, the two atoms must be in different molecules. For the *intra* setting, the two atoms must be in the same molecule.

If the *kspace* keyword is set to *yes*, which is not the default, and if a *kspace_style* is defined, then the interaction energy will include a Kspace component which is the long-range Coulombic energy between all the atoms in the first group and all the atoms in the second group. Likewise, the interaction force calculated by this compute will include the force on the compute group atoms due to long-range Coulombic interactions with atoms in the specified group2.

Normally the long-range Coulombic energy converges only when the net charge of the unit cell is zero. However, one can assume the net charge of the system is neutralized by a uniform background plasma, and a correction to the system energy can be applied to reduce artifacts. For more information see *(Bogusz)*. If the *boundary* keyword is set to *yes*, which is the default, and *kspace* contributions are included, then this energy correction term will be added to the total group-group energy. This correction term does not affect the force calculation and will be zero if one or both of the groups are charge neutral. This energy correction term is the same as that included in the regular Ewald and PPPM routines.

Note

The *molecule* setting only affects the group/group contributions calculated by the *pair* keyword. It does not affect the group/group contributions calculated by the *kspace* keyword.

This compute does not calculate any bond or angle or dihedral or improper interactions between atoms in the two groups.

The pairwise contributions to the group-group interactions are calculated by looping over a neighbor list. The Kspace contribution to the group-group interactions require essentially the same amount of work (FFTs, Ewald summation) as computing long-range forces for the entire system. Thus it can be costly to invoke this compute too frequently.

Note

If you have a bonded system, then the settings of *special_bonds* command can remove pairwise interactions between atoms in the same bond, angle, or dihedral. This is the default setting for the *special_bonds* command, and means those pairwise interactions do not appear in the neighbor list. Because this compute uses a neighbor list, it also means those pairs will not be included in the group/group interaction. This does not apply when using long-range Coulomb interactions (*coul/long*, *coul/msm*, *coul/wolf* or similar). One way to get around this would be to set *special_bond* scaling factors to very tiny numbers that are not exactly zero (e.g., 1.0×10^{-50}). Another workaround would be to write a dump file and use the *rerun* command to compute the group/group interactions for snapshots in the dump file. The rerun script can use a *special_bonds* command that includes all pairs in the neighbor list.

If you desire a breakdown of the interactions into a pairwise and Kspace component, simply invoke the compute twice with the appropriate yes/no settings for the *pair* and *kspace* keywords. This is no more costly than using a single compute with both keywords set to *yes*. The individual contributions can be summed in a *variable* if desired.

This document describes how the long-range group-group calculations are performed.

3.50.4 Output info

This compute calculates a global scalar (the energy) and a global vector of length 3 (force), which can be accessed by indices 1–3. These values can be used by any command that uses global scalar or vector values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

Both the scalar and vector values calculated by this compute are "extensive". The scalar value will be in energy *units*. The vector values will be in force *units*.

3.50.5 Restrictions

Not all pair styles can be evaluated in a pairwise mode as required by this compute. For example, three-body and other many-body potentials, such as *Tersoff* and *Stillinger-Weber* cannot be used. *EAM* potentials will re-use previously computed embedding term contributions, so the computed pairwise forces and energies are based on the whole system and not valid if particles have been moved since.

Not all *Kspace styles* support the calculation of group/group interactions. The regular *ewald* and *pppm* styles do.

3.50.6 Related commands

none

3.50.7 Default

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

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

3.51 compute gyration command

3.51.1 Syntax

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

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

3.51.2 Examples

```
compute 1 molecule gyration
```

3.51.3 Description

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

R_g is a measure of the size of the group of atoms, and is computed as the square root of the R_g^2 value in this formula

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

where M is the total mass of the group, $r_{\rm cm}$ is the center-of-mass position of the group, and the sum is over all atoms in the group.

A R_g^2 tensor, stored as a 6-element vector, is also calculated by this compute. The formula for the components of the tensor is the same as the above formula, except that $(r_i - r_{\rm cm})^2$ is replaced by $(r_{i,x} - r_{{\rm cm},x}) \cdot (r_{i,y} - r_{{\rm cm},y})$ for the xy component, and so on. The six components of the vector are ordered xx, yy, zz, xy, xz, yz. Note that unlike the scalar R_g, each of the six values of the tensor is effectively a "squared" value, since the cross-terms may be negative and taking a square root would be invalid.

Note

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

3.51.4 Output info

This compute calculates a global scalar (R_g) and a global vector of length 6 (R_g^2 tensor), which can be accessed by indices 1–6. These values can be used by any command that uses a global scalar value or vector values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

The scalar and vector values calculated by this compute are "intensive". The scalar and vector values will be in distance and distance^2 *units*, respectively.

3.51.5 Restrictions

none

3.51.6 Related commands

compute gyration/chunk, *compute gyration/shape*

3.51.7 Default

none

3.52 compute gyration/chunk command

3.52.1 Syntax

```
compute ID group-ID gyration/chunk chunkID keyword value ...
```

- ID, group-ID are documented in *compute* command
- gyration/chunk = style name of this compute command
- chunkID = ID of *compute chunk/atom* command
- zero or more keyword/value pairs may be appended
- keyword = *tensor*

  tensor value = none

3.52.2 Examples

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

3.52.3 Description

Define a computation that calculates the radius of gyration R_g for multiple chunks of atoms.

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* and *Howto chunk* doc pages for details of how chunks can be defined and examples of how they can be used to measure properties of a system.

This compute calculates the radius of gyration R_g for each chunk, which includes all effects due to atoms passing through periodic boundaries.

R_g is a measure of the size of a chunk, and is computed by the formula

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

where M is the total mass of the chunk, $r_{\rm cm}$ is the center-of-mass position of the chunk, and the sum is over all atoms in the chunk.

Note that only atoms in the specified group contribute to the calculation. The *compute chunk/atom* command defines its own group; atoms will have a chunk ID = 0 if they are not in that group, signifying they are not assigned to a chunk, and will thus also not contribute to this calculation. You can specify the "all" group for this command if you simply want to include atoms with non-zero chunk IDs.

If the *tensor* keyword is specified, then the scalar R_g value is not calculated, but an R_g tensor is instead calculated for each chunk. The formula for the components of the tensor is the same as the above formula, except that $(r_i - r_{\rm cm})^2$ is

replaced by $(r_{i,x} - r_{\mathrm{cm},x}) \cdot (r_{i,y} - r_{\mathrm{cm},y})$ for the *xy* component, and so on. The six components of the tensor are ordered *xx*, *yy*, *zz*, *xy*, *xz*, *yz*.

Note

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

The simplest way to output the results of the compute gyration/chunk calculation to a file is to use the *fix ave/time* command, for example:

```
compute cc1 all chunk/atom molecule
compute myChunk all gyration/chunk cc1
fix 1 all ave/time 100 1 100 c_myChunk file tmp.out mode vector
```

3.52.4 Output info

This compute calculates a global vector if the *tensor* keyword is not specified and a global array if it is. The length of the vector or number of rows in the array = the number of chunks *Nchunk* as calculated by the specified *compute chunk/atom* command. If the *tensor* keyword is specified, the global array has six columns. The vector or array can be accessed by any command that uses global values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

All the vector or array values calculated by this compute are "intensive". The vector or array values will be in distance *units*, since they are the square root of values represented by the formula above.

3.52.5 Restrictions

none

3.52.6 Related commands

none

compute gyration

3.52.7 Default

none

3.53 compute gyration/shape command

3.53.1 Syntax

```
compute ID group-ID gyration/shape compute-ID
```

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

3.53.2 Examples

```
compute 1 molecule gyration/shape pe
```

3.53.3 Description

Define a computation that calculates the eigenvalues of the gyration tensor of a group of atoms and three shape parameters. The computation includes all effects due to atoms passing through periodic boundaries.

The three computed shape parameters are the asphericity, b, the acylindricity, c, and the relative shape anisotropy, k, viz.,

$$b = l_z - \frac{1}{2}(l_y + l_x)$$
$$c = l_y - l_x$$
$$k = \frac{3}{2}\frac{l_x^2 + l_y^2 + l_z^2}{(l_x + l_y + l_z)^2} - \frac{1}{2}$$

where $l_x \leq l_y \leq l_z$ are the three eigenvalues of the gyration tensor. A general description of these parameters is provided in *(Mattice)* while an application to polymer systems can be found in *(Theodorou)*. The asphericity is always non-negative and zero only when the three principal moments are equal. This zero condition is met when the distribution of particles is spherically symmetric (hence the name asphericity) but also whenever the particle distribution is symmetric with respect to the three coordinate axes (e.g., when the particles are distributed uniformly on a cube, tetrahedron or other Platonic solid). The acylindricity is always non-negative and zero only when the two principal moments are equal. This zero condition is met when the distribution of particles is cylindrically symmetric (hence the name, acylindricity), but also whenever the particle distribution is symmetric with respect to the two coordinate axes (e.g., when the particles are distributed uniformly on a regular prism). The relative shape anisotropy is bounded between zero (if all points are spherically symmetric) and one (if all points lie on a line).

Note

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

3.53.4 Output info

This compute calculates a global vector of length 6, which can be accessed by indices 1–6. The first three values are the eigenvalues of the gyration tensor followed by the asphericity, the acylindricity and the relative shape anisotropy. The computed values can be used by any command that uses global vector values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

The vector values calculated by this compute are "intensive". The first five vector values will be in distance2 *units* while the sixth one is dimensionless.

3.53.5 Restrictions

This compute is part of the EXTRA-COMPUTE package. It is only enabled if LAMMPS was built with that package. See the *Build package* page for more info.

3.53.6 Related commands

compute gyration

3.53.7 Default

none

(Mattice) Mattice, Suter, Conformational Theory of Large Molecules, Wiley, New York, 1994.

(Theodorou) Theodorou, Suter, Macromolecules, 18, 1206 (1985).

3.54 compute gyration/shape/chunk command

3.54.1 Syntax

```
compute ID group-ID gyration/shape/chunk compute-ID
```

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

3.54.2 Examples

```
compute 1 molecule gyration/shape/chunk pe
```

3.54.3 Description

Define a computation that calculates the eigenvalues of the gyration tensor and three shape parameters of multiple chunks of atoms. The computation includes all effects due to atoms passing through periodic boundaries.

The three computed shape parameters are the asphericity, b, the acylindricity, c, and the relative shape anisotropy, k, viz.,

$$b = l_z - \frac{1}{2}(l_y + l_x)$$
$$c = l_y - l_x$$
$$k = \frac{3}{2}\frac{l_x^2 + l_y^2 + l_z^2}{(l_x + l_y + l_z)^2} - \frac{1}{2}$$

where $l_x \leq l_y \leq l_z$ are the three eigenvalues of the gyration tensor. A general description of these parameters is provided in *(Mattice)* while an application to polymer systems can be found in *(Theodorou)*. The asphericity is always non-negative and zero only when the three principal moments are equal. This zero condition is met when the distribution of particles is spherically symmetric (hence the name asphericity) but also whenever the particle distribution is symmetric with respect to the three coordinate axes (e.g., when the particles are distributed uniformly on a cube, tetrahedron, or other Platonic solid). The acylindricity is always non-negative and zero only when the two principal moments are equal. This zero condition is met when the distribution of particles is cylindrically symmetric (hence the name, acylindricity), but also whenever the particle distribution is symmetric with respect to the two coordinate axes (e.g., when the particles are distributed uniformly on a regular prism). The relative shape anisotropy is bounded between 0 (if all points are spherically symmetric) and 1 (if all points lie on a line).

The tensor keyword must be specified in the compute gyration/chunk command.

Note

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

3.54.4 Output info

This compute calculates a global array with six columns, which can be accessed by indices 1–6. The first three columns are the eigenvalues of the gyration tensor followed by the asphericity, the acylindricity and the relative shape anisotropy. The computed values can be used by any command that uses global array values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

The array calculated by this compute is "intensive". The first five columns will be in distance^2 *units* while the sixth one is dimensionless.

3.54.5 Restrictions

This compute is part of the EXTRA-COMPUTE package. It is only enabled if LAMMPS was built with that package. See the *Build package* page for more info.

3.54.6 Related commands

compute gyration/chunk *compute gyration/shape*

3.54.7 Default

none

(Mattice) Mattice, Suter, Conformational Theory of Large Molecules, Wiley, New York, 1994.

(Theodorou) Theodorou, Suter, Macromolecules, 18, 1206 (1985).

3.55 compute heat/flux command

3.55.1 Syntax

```
compute ID group-ID heat/flux ke-ID pe-ID stress-ID
```

- ID, group-ID are documented in *compute* command
- heat/flux = style name of this compute command
- ke-ID = ID of a compute that calculates per-atom kinetic energy
- pe-ID = ID of a compute that calculates per-atom potential energy
- stress-ID = ID of a compute that calculates per-atom stress

3.55.2 Examples

```
compute myFlux all heat/flux myKE myPE myStress
```

3.55.3 Description

Define a computation that calculates the heat flux vector based on contributions from atoms in the specified group. This can be used by itself to measure the heat flux through a set of atoms (e.g., a region between two thermostatted reservoirs held at different temperatures), or to calculate a thermal conductivity using the equilibrium Green-Kubo formalism.

For other non-equilibrium ways to compute a thermal conductivity, see the *Howto kappa* doc page. These include use of the *fix thermal/conductivity* command for the Muller-Plathe method. Or the *fix heat* command which can add or subtract heat from groups of atoms.

The compute takes three arguments which are IDs of other *computes*. One calculates per-atom kinetic energy (*ke-ID*), one calculates per-atom potential energy (*pe-ID*), and the third calculates per-atom stress (*stress-ID*).

Note

These other computes should provide values for all the atoms in the group this compute specifies. That means the other computes could use the same group as this compute, or they can just use group "all" (or any group whose atoms are superset of the atoms in this compute's group). LAMMPS does not check for this.

In case of two-body interactions, the heat flux **J** is defined as

$$\begin{aligned}\mathbf{J} &= \frac{1}{V}\left[\sum_i e_i\mathbf{v}_i - \sum_i \mathbf{S}_i\mathbf{v}_i\right] \\ &= \frac{1}{V}\left[\sum_i e_i\mathbf{v}_i + \sum_{i<j}\left(\mathbf{F}_{ij}\cdot\mathbf{v}_j\right)\mathbf{r}_{ij}\right] \\ &= \frac{1}{V}\left[\sum_i e_i\mathbf{v}_i + \frac{1}{2}\sum_{i<j}\left(\mathbf{F}_{ij}\cdot\left(\mathbf{v}_i+\mathbf{v}_j\right)\right)\mathbf{r}_{ij}\right]\end{aligned}$$

e_i in the first term of the equation is the per-atom energy (potential and kinetic). This is calculated by the computes *ke-ID* and *pe-ID*. $\mathbf{S}_i$ in the second term is the per-atom stress tensor calculated by the compute *stress-ID*. See *compute stress/atom* and *compute centroid/stress/atom* for possible definitions of atomic stress $\mathbf{S}_i$ in the case of bonded and many-body interactions. The tensor multiplies $\mathbf{v}_i$ by a 3×3 matrix to yield a vector. Note that as discussed below, the $1/V$ scaling factor in the equation for **J** is **not** included in the calculation performed by these computes; you need to add it for a volume appropriate to the atoms included in the calculation.

Note

The *compute pe/atom* and *compute stress/atom* commands have options for which terms to include in their calculation (pair, bond, etc). The heat flux calculation will thus include exactly the same terms. Normally you should use *compute stress/atom virial* or *compute centroid/stress/atom virial* so as not to include a kinetic energy term in the heat flux.

Warning

The compute *heat/flux* has been reported to produce unphysical values for angle, dihedral, improper and constraint force contributions when used with *compute stress/atom*, as discussed in *(Surblys2019)*, *(Boone)* and *(Surblys2021)*. You are strongly advised to use *compute centroid/stress/atom*, which has been implemented specifically for such cases.

Warning

Due to an implementation detail, the y and z components of heat flux from *fix rigid* contribution when computed via *compute stress/atom* are highly unphysical and should not be used.

The Green–Kubo formulas relate the ensemble average of the auto-correlation of the heat flux **J** to the thermal conductivity κ:

$$\kappa = \frac{V}{k_B T^2}\int_0^\infty \langle J_x(0)J_x(t)\rangle \,\mathrm{d}t = \frac{V}{3k_B T^2}\int_0^\infty \langle \mathbf{J}(0)\cdot\mathbf{J}(t)\rangle \,\mathrm{d}t$$

The heat flux can be output every so many timesteps (e.g., via the *thermo_style custom* command). Then as a post-processing operation, an auto-correlation can be performed, its integral estimated, and the Green–Kubo formula above evaluated.

The *fix ave/correlate* command can calculate the auto-correlation. The trap() function in the *variable* command can calculate the integral.

An example LAMMPS input script for solid argon is appended below. The result should be an average conductivity ≈ 0.29 W/m · K.

3.55.4 Output info

This compute calculates a global vector of length 6. The first three components are the x, y, and z components of the full heat flux vector (i.e., J_x, J_y, and J_z). The next three components are the x, y, and z components of just the convective portion of the flux (i.e., the first term in the equation for **J**). Each component can be accessed by indices 1–6. These values can be used by any command that uses global vector values from a compute as input. See the *Howto output* documentation for an overview of LAMMPS output options.

The vector values calculated by this compute are "extensive", meaning they scale with the number of atoms in the simulation. They can be divided by the appropriate volume to get a flux, which would then be an "intensive" value, meaning independent of the number of atoms in the simulation. Note that if the compute group is "all", then the appropriate volume to divide by is the simulation box volume. However, if a group with a subset of atoms is used, it should be the volume containing those atoms.

The vector values will be in energy*velocity *units*. Once divided by a volume the units will be that of flux, namely energy/area/time *units*

3.55.5 Restrictions

none

3.55.6 Related commands

fix thermal/conductivity, *fix ave/correlate*, *variable*

3.55.7 Default

none

Example Input File

```
# Sample LAMMPS input script for thermal conductivity of solid Ar

units       real
variable    T equal 70
variable    V equal vol
variable    dt equal 4.0
```

(continues on next page)

(continued from previous page)

```
variable    p equal 200     # correlation length
variable    s equal 10      # sample interval
variable    d equal $p*$s   # dump interval

# convert from LAMMPS real units to SI

variable    kB equal 1.3806504e-23    # [J/K] Boltzmann
variable    kCal2J equal 4186.0/6.02214e23
variable    A2m equal 1.0e-10
variable    fs2s equal 1.0e-15
variable    convert equal ${kCal2J}*${kCal2J}/${fs2s}/${A2m}

# setup problem

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

# equilibration and thermalization

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

# thermal conductivity calculation, switch to NVE if desired

#unfix       NVT
#fix         NVE all nve

reset_timestep 0
compute      myKE all ke/atom
compute      myPE all pe/atom
compute      myStress all stress/atom NULL virial
compute      flux all heat/flux myKE myPE myStress
variable     Jx equal c_flux[1]/vol
variable     Jy equal c_flux[2]/vol
variable     Jz equal c_flux[3]/vol
fix          JJ all ave/correlate $s $p $d &
             c_flux[1] c_flux[2] c_flux[3] type auto file J0Jt.dat ave running
variable     scale equal ${convert}/${kB}/$T/$T/$V*$s*${dt}
variable     k11 equal trap(f_JJ[3])*${scale}
variable     k22 equal trap(f_JJ[4])*${scale}
variable     k33 equal trap(f_JJ[5])*${scale}
thermo_style custom step temp v_Jx v_Jy v_Jz v_k11 v_k22 v_k33
```

(continues on next page)

(continued from previous page)

```
run         100000
variable    k equal (v_k11+v_k22+v_k33)/3.0
variable    ndens equal count(all)/vol
print       "average conductivity: $k[W/mK] @ $T K, ${ndens} /A\^3"
```

(Surblys2019) Surblys, Matsubara, Kikugawa, Ohara, Phys Rev E, 99, 051301(R) (2019).

(Boone) Boone, Babaei, Wilmer, J Chem Theory Comput, 15, 5579–5587 (2019).

(Surblys2021) Surblys, Matsubara, Kikugawa, Ohara, J Appl Phys 130, 215104 (2021).

3.56 compute hexorder/atom command

3.56.1 Syntax

```
compute ID group-ID hexorder/atom keyword values ...
```

- ID, group-ID are documented in *compute* command
- hexorder/atom = style name of this compute command
- one or more keyword/value pairs may be appended

  keyword = degree or nnn or cutoff
    cutoff value = distance cutoff
    nnn value = number of nearest neighbors
    degree value = degree n of order parameter

3.56.2 Examples

```
compute 1 all hexorder/atom
compute 1 all hexorder/atom degree 4 nnn 4 cutoff 1.2
```

3.56.3 Description

Define a computation that calculates q_n the bond-orientational order parameter for each atom in a group. The hexatic (n = 6) order parameter was introduced by *Nelson and Halperin* as a way to detect hexagonal symmetry in two-dimensional systems. For each atom, q_n is a complex number (stored as two real numbers) defined as follows:

$$q_n = \frac{1}{nnn} \sum_{j=1}^{nnn} e^{ni\theta(\mathbf{r}_{ij})}$$

where the sum is over the nnn nearest neighbors of the central atom. The angle θ is formed by the bond vector r_{ij} and the x axis. θ is calculated only using the x and y components, whereas the distance from the central atom is calculated using all three x, y, and z components of the bond vector. Neighbor atoms not in the group are included in the order parameter of atoms in the group.

The optional keyword *cutoff* defines the distance cutoff used when searching for neighbors. The default value, also the maximum allowable value, is the cutoff specified by the pair style.

The optional keyword *nnn* defines the number of nearest neighbors used to calculate q_n. The default value is 6. If the value is NULL, then all neighbors up to the distance cutoff are used.

The optional keyword *degree* sets the degree n of the order parameter. The default value is 6. For a perfect hexagonal lattice with *nnn* = 6, $q_6 = e^{6i\phi}$ for all atoms, where the constant $0 < \phi < \frac{\pi}{3}$ depends only on the orientation of the lattice relative to the x axis. In an isotropic liquid, local neighborhoods may still exhibit weak hexagonal symmetry, but because the orientational correlation decays quickly with distance, the value of ϕ will be different for different atoms, and so when q_6 is averaged over all the atoms in the system, $|\langle q_6 \rangle| << 1$.

The value of q_n is set to zero for atoms not in the specified compute group, as well as for atoms that have less than *nnn* neighbors within the distance cutoff.

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

Note

If you have a bonded system, then the settings of *special_bonds* command can remove pairwise interactions between atoms in the same bond, angle, or dihedral. This is the default setting for the *special_bonds* command, and means those pairwise interactions do not appear in the neighbor list. Because this fix uses the neighbor list, it also means those pairs will not be included in the order parameter. This difficulty can be circumvented by writing a dump file, and using the *rerun* command to compute the order parameter for snapshots in the dump file. The rerun script can use a *special_bonds* command that includes all pairs in the neighbor list.

3.56.4 Output info

This compute calculates a per-atom array with 2 columns, giving the real and imaginary parts q_n, a complex number restricted to the unit disk of the complex plane (i.e., $\Re(q_n)^2 + \Im(q_n)^2 \leq 1$).

These values can be accessed by any command that uses per-atom values from a compute as input. See the *Howto output* doc page for an overview of LAMMPS output options.

3.56.5 Restrictions

This compute is part of the EXTRA-COMPUTE package. It is only enabled if LAMMPS was built with that package. See the *Build package* page for more info.

3.56.6 Related commands

compute orientorder/atom, *compute coord/atom*, *compute centro/atom*

3.56.7 Default

The option defaults are *cutoff* = pair style cutoff, *nnn* = 6, *degree* = 6

(Nelson) Nelson, Halperin, Phys Rev B, 19, 2457 (1979).

3.57 compute hma command

3.57.1 Syntax

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

- ID, group-ID are documented in *compute* command
- hma = style name of this compute command
- temp-ID = ID of fix that specifies the set temperature during canonical simulation
- one or more keywords or keyword/argument pairs must be appended
- keyword = *anharmonic* or *u* or *p* or *cv*

anharmonic = compute will return anharmonic property values
u = compute will return potential energy
p value = Pharm = compute will return pressure
  Pharm = difference between the harmonic pressure and lattice pressure
    as described below
cv = compute will return the heat capacity

3.57.2 Examples

```
compute 2 all hma 1 u
compute 2 all hma 1 anharmonic u p 0.9
compute 2 all hma 1 u cv
```

3.57.3 Description

Define a computation that calculates the properties of a solid (potential energy, pressure or heat capacity), using the harmonically-mapped averaging (HMA) method. This command yields much higher precision than the equivalent compute commands (*compute pe*, *compute pressure*, etc.) commands during a canonical simulation of an atomic crystal. Specifically, near melting HMA can yield averages of a given precision an order of magnitude faster than conventional methods, and this only improves as the temperatures is lowered. This is particularly important for evaluating the free energy by thermodynamic integration, where the low-temperature contributions are the greatest source of statistical uncertainty. Moreover, HMA has other advantages, including smaller potential-truncation effects, finite-size effects, smaller timestep inaccuracy, faster equilibration and shorter decorrelation time.

HMA should not be used if atoms are expected to diffuse. It is also restricted to simulations in the NVT ensemble. While this compute may be used with any potential in LAMMPS, it will provide inaccurate results for potentials that do not go to 0 at the truncation distance; *pair_style lj/smooth/linear* and Ewald summation should work fine, while *pair_style lj/cut* will perform poorly unless the potential is shifted (via *pair_modify* shift) or the cutoff is large. Furthermore, computation of the heat capacity with this compute is restricted to those that implement the *single_hessian* method in Pair. Implementing *single_hessian* in additional pair styles is simple. Please contact Andrew Schultz (ajs42 at buffalo.edu) and David Kofke (kofke at buffalo.edu) if your desired pair style does not have this method. This is the list of pair styles that currently implement *single_hessian*:

- *pair_style lj/smooth/linear*

In this method, the analytically known harmonic behavior of a crystal is removed from the traditional ensemble averages, which leads to an accurate and precise measurement of the anharmonic contributions without contamination by noise

produced by the already-known harmonic behavior. A detailed description of this method can be found in (*Moustafa*). The potential energy is computed by the formula:

$$\langle U \rangle_{\mathrm{HMA}} = \frac{d}{2}(N-1)k_B T + \left\langle U + \frac{1}{2}\vec{F}\cdot\Delta\vec{r} \right\rangle$$

where N is the number of atoms in the system, k_B is Boltzmann's constant, T is the temperature, d is the dimensionality of the system (2 or 3 for 2d/3d), $\vec{F}\cdot\Delta\vec{r}$ is the sum of dot products of the atomic force vectors and displacement (from lattice sites) vectors, and U is the sum of pair, bond, angle, dihedral, improper, kspace (long-range), and fix energies.

The pressure is computed by the formula:

$$\langle P \rangle_{HMA} = \Delta\hat{P} + \left\langle P_{\mathrm{vir}} + \frac{\beta\Delta\hat{P}-\rho}{d(N-1)}\vec{F}\cdot\Delta\vec{r} \right\rangle$$

where ρ is the number density of the system, $\Delta\hat{P}$ is the difference between the harmonic and lattice pressure, P_{vir} is the virial pressure computed as the sum of pair, bond, angle, dihedral, improper, kspace (long-range), and fix contributions to the force on each atom, and $k_B = 1/k_B T$. Although the method will work for any value of $\Delta\hat{P}$ specified (use pressure *units*), the precision of the resultant pressure is sensitive to $\Delta\hat{P}$; the precision tends to be best when $\Delta\hat{P}$ is the actual the difference between the lattice pressure and harmonic pressure.

$$\langle C_V \rangle_{\mathrm{HMA}} = \frac{d}{2}(N-1)k_B + \frac{1}{k_B T^2}\left(\left\langle U_{\mathrm{HMA}}^2 \right\rangle - \langle U_{\mathrm{HMA}} \rangle^2\right) + \frac{1}{4T}\left\langle \vec{F}\cdot\Delta\vec{r} + \Delta r\cdot\Phi\cdot\Delta\vec{r} \right\rangle$$

where Φ is the Hessian matrix. The compute hma command computes the full expression for C_V except for the $\langle U_{\mathrm{HMA}} \rangle^2$ in the variance term, which can be obtained by passing the *u* keyword; you must add this extra contribution to the C_V value reported by this compute. The variance term can cause significant round-off error when computing C_V. To address this, the *anharmonic* keyword can be passed and/or the output format can be specified with more digits.

```
thermo_modify format float '%22.15e'
```

The *anharmonic* keyword will instruct the compute to return anharmonic properties rather than the full properties, which include lattice, harmonic and anharmonic contributions. When using this keyword, the compute must be first active (it must be included via a *thermo_style custom* command) while the atoms are still at their lattice sites (before equilibration).

The temp-ID specified with compute hma command should be same as the fix-ID of the Nose–Hoover (*fix nvt*) or Berendsen (*fix temp/berendsen*) thermostat used for the simulation. While using this command, the Langevin thermostat (*fix langevin*) should be avoided as its extra forces interfere with the HMA implementation.

Note

Compute hma command should be used right after the energy minimization, when the atoms are at their lattice sites. The simulation should not be started before this command has been used in the input script.

The following example illustrates the placement of this command in the input script:

```
min_style cg
minimize 1e-35 1e-15 50000 500000
compute 1 all hma thermostatid u
fix thermostatid all nvt temp 600.0 600.0 100.0
```

Note

Compute hma should be used when the atoms of the solid do not diffuse. Diffusion will reduce the precision in the potential energy computation.

Note

The *fix_modify energy yes* command must also be specified if a fix is to contribute potential energy to this command.

An example input script that uses this compute is included in examples/PACKAGES/hma/ along with corresponding LAMMPS output showing that the HMA properties fluctuate less than the corresponding conventional properties.

3.57.4 Output info

This compute calculates a global vector that includes the n properties requested as arguments to the command (the potential energy, pressure and/or heat capacity). The elements of the vector can be accessed by indices 1–n by any command that uses global vector values as input. See the *Howto output* page for an overview of LAMMPS output options.

The vector values calculated by this compute are "extensive". The scalar value will be in energy *units*.

3.57.5 Restrictions

This compute is part of the EXTRA-COMPUTE package. It is enabled only if LAMMPS was built with that package. See the *Build package* page for more info.

Usage restricted to canonical (NVT) ensemble simulation only.

3.57.6 Related commands

compute pe, *compute pressure*

dynamical matrix provides a finite difference formulation of the Hessian provided by Pair's single_hessian, which is used by this compute.

3.57.7 Default

none

(Moustafa) Sabry G. Moustafa, Andrew J. Schultz, and David A. Kofke, *Very fast averaging of thermal properties of crystals by molecular simulation*, Phys. Rev. E [92], 043303 (2015)

3.58 compute improper command

3.58.1 Syntax

```
compute ID group-ID improper
```

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

3.58.2 Examples

```
compute 1 all improper
```

3.58.3 Description

Define a computation that extracts the improper energy calculated by each of the improper sub-styles used in the *improper_style hybrid* command. These values are made accessible for output or further processing by other commands. The group specified for this command is ignored.

This compute is useful when using *improper_style hybrid* if you want to know the portion of the total energy contributed by one or more of the hybrid sub-styles.

3.58.4 Output info

This compute calculates a global vector of length N, where N is the number of sub_styles defined by the *improper_style hybrid* command. These styles can be accessed by the indices 1 through N. These values can be used by any command that uses global scalar or vector values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

The vector values are "extensive" and will be in energy *units*.

3.58.5 Restrictions

none

3.58.6 Related commands

compute pe, *compute pair*

3.58.7 Default

none

3.59 compute improper/local command

3.59.1 Syntax

```
compute ID group-ID improper/local value1 value2 ...
```

- ID, group-ID are documented in *compute* command
- improper/local = style name of this compute command
- one or more values may be appended
- value = *chi*

  chi = tabulate improper angles

3.59.2 Examples

```
compute 1 all improper/local chi
```

3.59.3 Description

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

The value *chi* is the improper angle, as defined in the doc pages for the individual improper styles listed on *improper_style* doc page.

The local data stored by this command is generated by looping over all the atoms owned on a processor and their impropers. An improper will only be included if all four atoms in the improper are in the specified compute group.

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

Here is an example of how to do this:

```
compute 1 all property/local itype iatom1 iatom2 iatom3 iatom4
compute 2 all improper/local chi
dump 1 all local 1000 tmp.dump index c_1[1] c_1[2] c_1[3] c_1[4] c_1[5] c_2[1]
```

3.59.4 Output info

This compute calculates a local vector or local array depending on the number of keywords. The length of the vector or number of rows in the array is the number of impropers. If a single keyword is specified, a local vector is produced. If two or more keywords are specified, a local array is produced where the number of columns = the number of keywords. The vector or array can be accessed by any command that uses local values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

The output for *chi* will be in degrees.

3.59.5 Restrictions

none

3.59.6 Related commands

dump local, *compute property/local*

3.59.7 Default

none

3.60 compute inertia/chunk command

3.60.1 Syntax

```
compute ID group-ID inertia/chunk chunkID
```

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

3.60.2 Examples

```
compute 1 fluid inertia/chunk molchunk
```

3.60.3 Description

Define a computation that calculates the inertia tensor for multiple chunks of atoms.

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* and *Howto chunk* doc pages for details of how chunks can be defined and examples of how they can be used to measure properties of a system.

This compute calculates the six components of the symmetric inertia tensor for each chunk, ordered $I_{xx}, I_{yy}, I_{zz}, I_{xy}, I_{yz}, I_{xz}$. The calculation includes all effects due to atoms passing through periodic boundaries.

Note that only atoms in the specified group contribute to the calculation. The *compute chunk/atom* command defines its own group; atoms will have a chunk ID = 0 if they are not in that group, signifying they are not assigned to a chunk, and will thus also not contribute to this calculation. You can specify the "all" group for this command if you simply want to include atoms with non-zero chunk IDs.

Note

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

The simplest way to output the results of the compute inertia/chunk calculation to a file is to use the *fix ave/time* command, for example:

```
compute cc1 all chunk/atom molecule
compute myChunk all inertia/chunk cc1
fix 1 all ave/time 100 1 100 c_myChunk[*] file tmp.out mode vector
```

3.60.4 Output info

This compute calculates a global array where the number of rows = the number of chunks *Nchunk* as calculated by the specified *compute chunk/atom* command. The number of columns is 6, one for each of the 6 components of the inertia tensor for each chunk, ordered as listed above. These values can be accessed by any command that uses global array values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

The array values are "intensive". The array values will be in mass*distance^2 *units*.

3.60.5 Restrictions

none

3.60.6 Related commands

variable inertia() function

3.60.7 Default

none

3.61 compute ke command

3.61.1 Syntax

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

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

3.61.2 Examples

```
compute 1 all ke
```

3.61.3 Description

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

The kinetic energy of each particle is computed as $\frac{1}{2}mv^2$, where m and v are the mass and velocity of the particle, respectively.

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

3.61.4 Output info

This compute calculates a global scalar (the summed KE). This value can be used by any command that uses a global scalar value from a compute as input. See the *Howto output* doc page for an overview of LAMMPS output options.

The scalar value calculated by this compute is "extensive". The scalar value will be in energy *units*.

3.61.5 Restrictions

none

3.61.6 Related commands

compute erotate/sphere

3.61.7 Default

none

3.62 compute ke/atom command

3.62.1 Syntax

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

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

3.62.2 Examples

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

3.62.3 Description

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

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

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

3.62.4 Output info

This compute calculates a per-atom vector, which can be accessed by any command that uses per-atom values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

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

3.62.5 Restrictions

none

3.62.6 Related commands

dump custom

3.62.7 Default

none

3.63 compute ke/atom/eff command

3.63.1 Syntax

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

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

3.63.2 Examples

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

3.63.3 Description

Define a computation that calculates the per-atom translational (nuclei and electrons) and radial kinetic energy (electron only) in a group. The particles are assumed to be nuclei and electrons modeled with the *electronic force field*.

The kinetic energy for each nucleus is computed as $\frac{1}{2}mv^2$, where m corresponds to the corresponding nuclear mass, and the kinetic energy for each electron is computed as $\frac{1}{2}(m_e v^2 + \frac{3}{4}m_e s^2)$, where m_e and v correspond to the mass and translational velocity of each electron, and s to its radial velocity, respectively.

There is a subtle difference between the quantity calculated by this compute and the kinetic energy calculated by the *ke* or *etotal* keyword used in thermodynamic output, as specified by the *thermo_style* command. For this compute, kinetic energy is "translational" plus electronic "radial" kinetic energy, calculated by the simple formula above. For thermodynamic output, the *ke* keyword infers kinetic energy from the temperature of the system with $\frac{1}{2}k_B T$ of energy for each (nuclear-only) degree of freedom in eFF.

Note

The temperature in eFF should be monitored via the *compute temp/eff* command, which can be printed with thermodynamic output by using the *thermo_modify* command, as shown in the following example:

```
compute         effTemp all temp/eff
thermo_style    custom step etotal pe ke temp press
thermo_modify   temp effTemp
```

The value of the kinetic energy will be 0.0 for atoms (nuclei or electrons) not in the specified compute group.

3.63.4 Output info

This compute calculates a scalar quantity for each atom, which can be accessed by any command that uses per-atom computes as input. See the *Howto output* page for an overview of LAMMPS output options.

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

3.63.5 Restrictions

This compute 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.

3.63.6 Related commands

dump custom

3.63.7 Default

none

3.64 compute ke/eff command

3.64.1 Syntax

```
compute ID group-ID ke/eff
```

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

3.64.2 Examples

```
compute 1 all ke/eff
```

3.64.3 Description

Define a computation that calculates the kinetic energy of motion of a group of eFF particles (nuclei and electrons), as modeled with the *electronic force field*.

The kinetic energy for each nucleus is computed as $\frac{1}{2}mv^2$ and the kinetic energy for each electron is computed as $\frac{1}{2}(m_e v^2 + \frac{3}{4}m_e s^2)$, where m corresponds to the nuclear mass, m_e to the electron mass, v to the translational velocity of each particle, and s to the radial velocity of the electron, respectively.

There is a subtle difference between the quantity calculated by this compute and the kinetic energy calculated by the *ke* or *etotal* keyword used in thermodynamic output, as specified by the *thermo_style* command. For this compute, kinetic energy is "translational" and "radial" (only for electrons) kinetic energy, calculated by the simple formula above. For thermodynamic output, the *ke* keyword infers kinetic energy from the temperature of the system with $\frac{1}{2}k_B T$ of energy for each degree of freedom. For the eFF temperature computation via the *compute temp_eff* command, these are the same. But different computes that calculate temperature can subtract out different non-thermal components of velocity and/or include other degrees of freedom.

Warning

The temperature in eFF models should be monitored via the *compute temp/eff* command, which can be printed with thermodynamic output by using the *thermo_modify* command, as shown in the following example:

```
compute         effTemp all temp/eff
thermo_style    custom step etotal pe ke temp press
thermo_modify   temp effTemp
```

See *compute temp/eff*.

3.64.4 Output info

This compute calculates a global scalar (the KE). This value can be used by any command that uses a global scalar value from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

The scalar value calculated by this compute is "extensive". The scalar value will be in energy *units*.

3.64.5 Restrictions

This compute 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.

3.64.6 Related commands

none

3.64.7 Default

none

3.65 compute ke/rigid command

3.65.1 Syntax

```
compute ID group-ID ke/rigid fix-ID
```

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

3.65.2 Examples

```
compute 1 all ke/rigid myRigid
```

3.65.3 Description

Define a computation that calculates the translational kinetic energy of a collection of rigid bodies, as defined by one of the *fix rigid* command variants.

The kinetic energy of each rigid body is computed as $\frac{1}{2}MV_{\mathrm{cm}}^2$, where M is the total mass of the rigid body, and V_{cm} is its center-of-mass velocity.

The *fix-ID* should be the ID of one of the *fix rigid* commands which defines the rigid bodies. The group specified in the compute command is ignored. The kinetic energy of all the rigid bodies defined by the fix rigid command in included in the calculation.

3.65.4 Output info

This compute calculates a global scalar (the summed KE of all the rigid bodies). This value can be used by any command that uses a global scalar value from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

The scalar value calculated by this compute is "extensive". The scalar value will be in energy *units*.

3.65.5 Restrictions

This compute 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.

3.65.6 Related commands

compute erotate/rigid

3.65.7 Default

none

3.66 compute mliap command

3.66.1 Syntax

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

- ID, group-ID are documented in *compute* command
- mliap = style name of this compute command
- two or more keyword/value pairs must be appended
- keyword = *model* or *descriptor* or *gradgradflag*

  model values = style
    style = linear or quadratic or mliappy
  descriptor values = style filename
    style = sna or ace
    filename = name of file containing descriptor definitions
  gradgradflag value = 0/1
    toggle gradgrad method for force gradient

3.66.2 Examples

```
compute mliap model linear descriptor sna Ta06A.mliap.descriptor
compute mliap model linear descriptor ace H_N_O_ccs.yace gradgradflag 1
```

3.66.3 Description

Compute style *mliap* provides a general interface to the gradient of machine-learning interatomic potentials with respect to model parameters. It is used primarily for calculating the gradient of energy, force, and stress components with respect to model parameters, which is useful when training *mliap pair_style* models to match target data. It provides separate definitions of the interatomic potential functional form (*model*) and the geometric quantities that characterize the atomic positions (*descriptor*). By defining *model* and *descriptor* separately, it is possible to use many different models with a given descriptor, or many different descriptors with a given model. Currently, the compute supports

linear and *quadratic* SNAP descriptor computes used in *pair_style snap*, *linear* SO3 descriptor computes, and *linear* ACE descriptor computes used in *pair_style pace*, and it is straightforward to add new descriptor styles.

The compute *mliap* command must be followed by two keywords *model* and *descriptor* in either order.

The *model* keyword is followed by the model style (*linear*, *quadratic* or *mliappy*). The *mliappy* model is only available if LAMMPS is built with the *mliappy* Python module. There are *specific installation instructions* for that module. For the *mliap* compute, specifying a *linear* model will compute the specified descriptors and gradients with respect to linear model parameters whereas *quadratic* will do the same, but for the quadratic products of descriptors.

The *descriptor* keyword is followed by a descriptor style, and additional arguments. The compute currently supports three descriptor styles: *sna*, *so3*, and *ace*, but it is is straightforward to add additional descriptor styles. The SNAP descriptor style *sna* is the same as that used by *pair_style snap*, including the linear, quadratic, and chem variants. A single additional argument specifies the descriptor filename containing the parameters and setting used by the SNAP descriptor. The descriptor filename usually ends in the *.mliap.descriptor* extension. The format of this file is identical to the descriptor file in the *pair_style mliap*, and is described in detail there.

The ACE descriptor style *ace* is the same as *pair_style pace*. A single additional argument specifies the *ace* descriptor filename that contains parameters and settings for the ACE descriptors. This file format differs from the SNAP or SO3 descriptor files, and has a *.yace* or *.ace* extension. However, as with other mliap descriptor styles, this file is identical to the ace descriptor file in *pair_style mliap*, where it is described in further detail.

Note

The number of LAMMPS atom types (and the value of *nelems* in the model) must match the value of *nelems* in the descriptor file.

Compute *mliap* calculates a global array containing gradient information. The number of columns in the array is *nelems* $\times$ *nparams* + 1. The first row of the array contain the derivative of potential energy with respect to. to each parameter and each element. The last six rows of the array contain the corresponding derivatives of the virial stress tensor, listed in Voigt notation: *pxx*, *pyy*, *pzz*, *pyz*, *pxz*, and *pxy*. In between the energy and stress rows are the 3*N* rows containing the derivatives of the force components. See section below on output for a detailed description of how rows and columns are ordered.

The element in the last column of each row contains the potential energy, force, or stress, according to the row. These quantities correspond to the user-specified reference potential that must be subtracted from the target data when training a model. The potential energy calculation uses the built in compute *thermo_pe*. The stress calculation uses a compute called *mliap_press* that is automatically created behind the scenes, according to the following command:

```
compute mliap_press all pressure NULL virial
```

See section below on output for a detailed explanation of the data layout in the global array.

The optional keyword *gradgradflag* controls how the force gradient is calculated. A value of 1 requires that the model provide the matrix of double gradients of energy with respect to both parameters and descriptors. For the linear and quadratic models this matrix is sparse and so is easily calculated and stored. For other models, this matrix may be prohibitively expensive to calculate and store. A value of 0 requires that the descriptor provide the derivative of the descriptors with respect to the position of every neighbor atom. This is not optimal for linear and quadratic models, but may be a better choice for more complex models.

Atoms not in the group do not contribute to this compute. Neighbor atoms not in the group do not contribute to this compute. The neighbor list needed to compute this quantity is constructed each time the calculation is performed (i.e., each time a snapshot of atoms is dumped). Thus it can be inefficient to compute/dump this quantity too frequently.

Note

If the user-specified reference potentials includes bonded and non-bonded pairwise interactions, then the settings of *special_bonds* command can remove pairwise interactions between atoms in the same bond, angle, or dihedral. This is the default setting for the *special_bonds* command, and means those pairwise interactions do not appear in the neighbor list. Because this fix uses the neighbor list, it also means those pairs will not be included in the calculation. The *rerun* command is not an option here, since the reference potential is required for the last column of the global array. A work-around is to prevent pairwise interactions from being removed by explicitly adding a *tiny* positive value for every pairwise interaction that would otherwise be set to zero in the *special_bonds* command.

3.66.4 Output info

Compute *mliap* evaluates a global array. The columns are arranged into *nelems* blocks, listed in order of element *I*. Each block contains one column for each of the *nparams* model parameters. A final column contains the corresponding energy, force component on an atom, or virial stress component. The rows of the array appear in the following order:

- 1 row: Derivatives of potential energy with respect to each parameter of each element.
- 3*N* rows: Derivatives of force components; the *x*, *y*, and *z* components of the force on atom *i* appear in consecutive rows. The atoms are sorted based on atom ID.
- 6 rows: Derivatives of the virial stress tensor with respect to each parameter of each element. The ordering of the rows follows Voigt notation: *pxx*, *pyy*, *pzz*, *pyz*, *pxz*, *pxy*.

These values can be accessed by any command that uses a global array from a compute as input. See the *Howto output* doc page for an overview of LAMMPS output options. To see how this command can be used within a Python workflow to train machine-learning interatomic potentials, see the examples in FitSNAP.

3.66.5 Restrictions

This compute is part of the ML-IAP package. It is only enabled if LAMMPS was built with that package. In addition, building LAMMPS with the ML-IAP package requires building LAMMPS with the ML-SNAP package. The *mliappy* model also requires building LAMMPS with the PYTHON package. The *ace* descriptor also requires building LAMMPS with the ML-PACE package. See the *Build package* page for more info. Note that *kk* (KOKKOS) accelerated variants of SNAP and ACE descriptors are not compatible with *mliap descriptor*.

3.66.6 Related commands

pair_style mliap

3.66.7 Default

The keyword defaults are gradgradflag = 1

3.67 compute momentum command

3.67.1 Syntax

```
compute ID group-ID momentum
```

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

3.67.2 Examples

```
compute 1 all momentum
```

3.67.3 Description

Define a computation that calculates the translational momentum p of a group of particles. It is computed as the sum $\vec{p} = \sum_i m_i \cdot \vec{v}_i$ over all particles in the compute group, where m and v are the mass and velocity vector of the particle, respectively.

3.67.4 Output info

This compute calculates a global vector (the summed momentum) of length 3. This value can be used by any command that uses a global vector value from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

The vector value calculated by this compute is "extensive". The vector value will be in mass*velocity *units*.

3.67.5 Restrictions

This compute is part of the EXTRA-COMPUTE package. It is only enabled if LAMMPS was built with that package. See the *Build package* page for more info.

3.67.6 Related commands

3.67.7 Default

none

3.68 compute msd command

3.68.1 Syntax

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

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

  com value = yes or no
  average value = yes or no

3.68.2 Examples

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

3.68.3 Description

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

A vector of four quantities is calculated by this compute. The first three elements of the vector are the squared *dx*, *dy*, and *dz* displacements, summed and averaged over atoms in the group. The fourth element is the total squared displacement (i.e., $dx^2 + dy^2 + dz^2$), summed and averaged over atoms in the group.

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

The displacement of an atom is from its reference position. This is normally the original position at the time the compute command was issued, unless the *average* keyword is set to *yes*. The value of the displacement will be 0.0 for atoms not in the specified compute group.

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

If the *average* option is set to *yes* then the reference position of an atom is based on the average position of that atom, corrected for center-of-mass motion if requested. The average position is a running average over all previous calls to the compute, including the current call. So on the first call it is current position, on the second call it is the arithmetic average of the current position and the position on the first call, and so on. Note that when using this option, the precise value of the mean square displacement will depend on the number of times the compute is called. So, for example, changing the frequency of thermo output may change the computed displacement. Also, the precise values will be changed if a single simulation is broken up into two parts, using either multiple run commands or a restart file. It only makes sense to use this option if the atoms are not diffusing, so that their average positions relative to the center of mass of the system are stationary. The most common case is crystalline solids undergoing thermal motion.

Note

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

Note

If you want the quantities calculated by this compute to be continuous when running from a *restart file*, then you should use the same ID for this compute, as in the original run. This is so that the fix this compute creates to store per-atom quantities will also have the same ID, and thus be initialized correctly with atom reference positions from the restart file. When *average* is set to yes, then the atom reference positions are restored correctly, but not the number of samples used obtain them. As a result, the reference positions from the restart file are combined with subsequent positions as if they were from a single sample, instead of many, which will change the values of msd somewhat.

3.68.4 Output info

This compute calculates a global vector of length 4, which can be accessed by indices 1–4 by any command that uses global vector values from a compute as input. See the *Howto output* doc page for an overview of LAMMPS output options.

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

3.68.5 Restrictions

Compute *msd* cannot be used with a dynamic group.

3.68.6 Related commands

compute msd/nongauss, *compute displace_atom*, *fix store/state*, *compute msd/chunk*

3.68.7 Default

The option default are com = no, average = no.

3.69 compute msd/chunk command

3.69.1 Syntax

```
compute ID group-ID msd/chunk chunkID
```

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

- chunkID = ID of *compute chunk/atom* command

3.69.2 Examples

```
compute 1 all msd/chunk molchunk
```

3.69.3 Description

Define a computation that calculates the mean-squared displacement (MSD) for multiple chunks of atoms.

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* and *Howto chunk* doc pages for details of how chunks can be defined and examples of how they can be used to measure properties of a system.

Four quantities are calculated by this compute for each chunk. The first 3 quantities are the squared dx, dy, and dz displacements of the center-of-mass. The fourth component is the total squared displacement (i.e., $dx^2 + dy^2 + dz^2$) of the center-of-mass. These calculations include all effects due to atoms passing through periodic boundaries.

Note that only atoms in the specified group contribute to the calculation. The *compute chunk/atom* command defines its own group; atoms will have a chunk ID = 0 if they are not in that group, signifying they are not assigned to a chunk, and will thus also not contribute to this calculation. You can specify the "all" group for this command if you simply want to include atoms with non-zero chunk IDs.

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

The displacement of the center-of-mass of the chunk is from its original center-of-mass position, calculated on the timestep this compute command was first invoked.

Note

The number of chunks *Nchunk* calculated by the *compute chunk/atom* command must remain constant each time this compute is invoked, so that the displacement for each chunk from its original position can be computed consistently. If *Nchunk* does not remain constant, an error will be generated. If needed, you can enforce a constant *Nchunk* by using the *nchunk once* or *ids once* options when specifying the *compute chunk/atom* command.

Note

This compute stores the original position (of the center-of-mass) of each chunk. When a displacement is calculated on a later timestep, it is assumed that the same atoms are assigned to the same chunk ID. However LAMMPS has no simple way to ensure this is the case, though you can use the *ids once* option when specifying the *compute chunk/atom* command. Note that if this is not the case, the MSD calculation does not have a sensible meaning.

Note

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

Note

If you want the quantities calculated by this compute to be continuous when running from a *restart file*, then you should use the same ID for this compute, as in the original run. This is so that the fix this compute creates to store per-chunk quantities will also have the same ID, and thus be initialized correctly with chunk reference positions from the restart file.

The simplest way to output the results of the compute msd/chunk calculation to a file is to use the *fix ave/time* command, for example:

```
compute cc1 all chunk/atom molecule
compute myChunk all msd/chunk cc1
fix 1 all ave/time 100 1 100 c_myChunk[*] file tmp.out mode vector
```

3.69.4 Output info

This compute calculates a global array where the number of rows = the number of chunks *Nchunk* as calculated by the specified *compute chunk/atom* command. The number of columns = 4 for *dx*, *dy*, *dz*, and the total displacement. These values can be accessed by any command that uses global array values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

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

3.69.5 Restrictions

none

3.69.6 Related commands

compute msd

3.69.7 Default

none

3.70 compute msd/nongauss command

3.70.1 Syntax

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

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

  com value = yes or no

3.70.2 Examples

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

3.70.3 Description

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

A vector of three quantities is calculated by this compute. The first element of the vector is the total squared displacement, $dr^2 = dx^2 + dy^2 + dz^2$, of the atoms, and the second is the fourth power of these displacements, $dr^4 = (dx^2 + dy^2 + dz^2)^2$, summed and averaged over atoms in the group. The third component is the non-Gaussian diffusion parameter NGP,

$$\mathrm{NGP}(t) = \frac{3\left\langle (r(t) - r(0))^4 \right\rangle}{5\left\langle (r(t) - r(0))^2 \right\rangle^2} - 1.$$

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

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

See the *compute msd* page for further important NOTEs, which also apply to this compute.

3.70.4 Output info

This compute calculates a global vector of length 3, which can be accessed by indices 1–3 by any command that uses global vector values from a compute as input. See the *Howto output* doc page for an overview of LAMMPS output options.

The vector values are "intensive". The first vector value will be in distance^2 *units*, the second is in distance^4 units, and the third is dimensionless.

3.70.5 Restrictions

Compute *msd/nongauss* cannot be used with a dynamic group.

This compute is part of the EXTRA-COMPUTE package. It is only enabled if LAMMPS was built with that package. See the *Build package* page for more info.

3.70.6 Related commands

compute msd

3.70.7 Default

The option default is com = no.

3.71 compute nbond/atom command

3.71.1 Syntax

```
compute ID group-ID nbond/atom keyword value
```

- ID, group-ID are documented in *compute* command
- nbond/atom = style name of this compute command
- zero or more keyword/value pairs may be appended
- keyword = *bond/type*

  bond/type value = btype
    btype = bond type included in count

3.71.2 Examples

```
compute 1 all nbond/atom
compute 1 all nbond/atom bond/type 2
```

3.71.3 Description

Added in version 4May2022.

Define a computation that computes the number of bonds each atom is part of. Bonds which are broken are not counted in the tally. See the *Howto broken bonds* page for more information. The number of bonds will be zero for atoms not in the specified compute group. This compute does not depend on Newton bond settings.

If the keyword *bond/type* is specified, only bonds of *btype* are counted.

3.71.4 Output info

This compute calculates a per-atom vector, which can be accessed by any command that uses per-atom values from a compute as input. See the *Howto output* doc page for an overview of LAMMPS output options.

3.71.5 Restrictions

This compute 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.

3.71.6 Related commands

3.71.7 Default

none

3.72 compute omega/chunk command

3.72.1 Syntax

```
compute ID group-ID omega/chunk chunkID
```

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

3.72.2 Examples

```
compute 1 fluid omega/chunk molchunk
```

3.72.3 Description

Define a computation that calculates the angular velocity (omega) of multiple chunks of atoms.

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* and *Howto chunk* doc pages for details of how chunks can be defined and examples of how they can be used to measure properties of a system.

This compute calculates the three components of the angular velocity vector for each chunk via the formula $\vec{L} = \mathrm{I} \cdot \vec{\omega}$, where $\vec{L}$ is the angular momentum vector of the chunk, I is its moment of inertia tensor, and ω is the angular velocity of the chunk. The calculation includes all effects due to atoms passing through periodic boundaries.

Note that only atoms in the specified group contribute to the calculation. The *compute chunk/atom* command defines its own group; atoms will have a chunk ID = 0 if they are not in that group, signifying they are not assigned to a chunk, and will thus also not contribute to this calculation. You can specify the "all" group for this command if you simply want to include atoms with non-zero chunk IDs.

Note

The coordinates of an atom contribute to the chunk's angular velocity in "unwrapped" form, by using the image flags associated with each atom. See the *dump custom* command for a discussion of "unwrapped" coordinates. See

the Atoms section of the *read_data* command for a discussion of image flags and how they are set for each atom. You can reset the image flags (e.g., to 0) before invoking this compute by using the *set image* command.

The simplest way to output the results of the compute omega/chunk calculation to a file is to use the *fix ave/time* command, for example:

```
compute cc1 all chunk/atom molecule
compute myChunk all omega/chunk cc1
fix 1 all ave/time 100 1 100 c_myChunk[*] file tmp.out mode vector
```

3.72.4 Output info

This compute calculates a global array where the number of rows is the number of chunks *Nchunk* as calculated by the specified *compute chunk/atom* command. The number of columns is 3 for the three (x, y, z) components of the angular velocity for each chunk. These values can be accessed by any command that uses global array values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

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

3.72.5 Restrictions

none

3.72.6 Related commands

variable omega() function

3.72.7 Default

none

3.73 compute orientorder/atom command

Accelerator Variants: *orientorder/atom/kk*

3.73.1 Syntax

```
compute ID group-ID orientorder/atom keyword values ...
```

- ID, group-ID are documented in *compute* command
- orientorder/atom = style name of this compute command
- one or more keyword/value pairs may be appended

keyword = cutoff or nnn or degrees or wl or wl/hat or components or chunksize
  cutoff value = distance cutoff
  nnn value = number of nearest neighbors
  degrees values = nlvalues, l1, l2,...
  wl value = yes or no
  wl/hat value = yes or no
  components value = ldegree
  chunksize value = number of atoms in each pass

3.73.2 Examples

```
compute 1 all orientorder/atom
compute 1 all orientorder/atom degrees 5 4 6 8 10 12 nnn NULL cutoff 1.5
compute 1 all orientorder/atom wl/hat yes
compute 1 all orientorder/atom components 6
```

3.73.3 Description

Define a computation that calculates a set of bond-orientational order parameters Q_ℓ for each atom in a group. These order parameters were introduced by *Steinhardt et al.* as a way to characterize the local orientational order in atomic structures. For each atom, Q_ℓ is a real number defined as follows:

$$\bar{Y}_{\ell m} = \frac{1}{nnn} \sum_{j=1}^{nnn} Y_{\ell m}\big(\theta(\mathbf{r}_{ij}), \phi(\mathbf{r}_{ij})\big)$$

$$Q_\ell = \sqrt{\frac{4\pi}{2\ell+1} \sum_{m=-\ell}^{m=\ell} \bar{Y}_{\ell m} \bar{Y}^*_{\ell m}}$$

The first equation defines the local order parameters as averages of the spherical harmonics $Y_{\ell m}$ for each neighbor. These are complex number components of the 3D analog of the 2D order parameter q_n, which is implemented as LAMMPS compute *hexorder/atom*. The summation is over the *nnn* nearest neighbors of the central atom. The angles θ and ϕ are the standard spherical polar angles defining the direction of the bond vector r_{ij}. The phase and sign of $Y_{\ell m}$ follow the standard conventions, so that sign($Y_{\ell\ell}(0,0)$) = $(-1)^\ell$. The second equation defines Q_ℓ, which is a rotationally invariant non-negative amplitude obtained by summing over all the components of degree ℓ.

The optional keyword *cutoff* defines the distance cutoff used when searching for neighbors. The default value, also the maximum allowable value, is the cutoff specified by the pair style.

The optional keyword *nnn* defines the number of nearest neighbors used to calculate Q_ℓ. The default value is 12. If the value is NULL, then all neighbors up to the specified distance cutoff are used.

The optional keyword *degrees* defines the list of order parameters to be computed. The first argument *nlvalues* is the number of order parameters. This is followed by that number of non-negative integers giving the degree of each order parameter. Because Q_2 and all odd-degree order parameters are zero for atoms in cubic crystals (see *Steinhardt*), the default order parameters are Q_4, Q_6, Q_8, Q_{10}, and Q_{12}. For the FCC crystal with *nnn* =12,

$$Q_4 = \sqrt{\frac{7}{192}} \approx 0.19094$$

The numerical values of all order parameters up to Q_{12} for a range of commonly encountered high-symmetry structures are given in Table I of *Mickel et al.*, and these can be reproduced with this compute.

The optional keyword *wl* will output the third-order invariants W_ℓ (see Eq. 1.4 in *Steinhardt*) for the same degrees as for the Q_ℓ parameters. For the FCC crystal with *nnn* = 12,

$$W_4 = -\sqrt{\frac{14}{143}}\left(\frac{49}{4096}\right)\pi^{-3/2} \approx -0.0006722136$$

The optional keyword *wl/hat* will output the normalized third-order invariants $\hat{W}_\ell$ (see Eq. 2.2 in *Steinhardt*) for the same degrees as for the Q_ℓ parameters. For the FCC crystal with *nnn* =12,

$$\hat{W}_4 = -\frac{7}{3}\sqrt{\frac{2}{429}} \approx -0.159317$$

The numerical values of $\hat{W}_\ell$ for a range of commonly encountered high-symmetry structures are given in Table I of *Steinhardt*, and these can be reproduced with this keyword.

The optional keyword *components* will output the components of the *normalized* complex vector $\hat{Y}_{\ell m} = \bar{Y}_{\ell m}/|\bar{Y}_{\ell m}|$ of degree *ldegree*, which must be included in the list of order parameters to be computed. This option can be used in conjunction with *compute coord_atom* to calculate the ten Wolde's criterion to identify crystal-like particles, as discussed in *ten Wolde*.

The optional keyword *chunksize* is only applicable when using the the KOKKOS package and is ignored otherwise. This keyword controls the number of atoms in each pass used to compute the bond-orientational order parameters and is used to avoid running out of memory. For example if there are 32768 atoms in the simulation and the *chunksize* is set to 16384, the parameter calculation will be broken up into two passes.

The value of Q_ℓ is set to zero for atoms not in the specified compute group, as well as for atoms that have less than *nnn* neighbors within the distance cutoff, unless *nnn* is NULL.

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

Note

If you have a bonded system, then the settings of *special_bonds* command can remove pairwise interactions between atoms in the same bond, angle, or dihedral. This is the default setting for the *special_bonds* command, and means those pairwise interactions do not appear in the neighbor list. Because this fix uses the neighbor list, it also means those pairs will not be included in the order parameter. This difficulty can be circumvented by writing a dump file, and using the *rerun* command to compute the order parameter for snapshots in the dump file. The rerun script can use a *special_bonds* command that includes all pairs in the neighbor list.

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.

3.73.4 Output info

This compute calculates a per-atom array with *nlvalues* columns, giving the Q_ℓ values for each atom, which are real numbers in the range $0 \leq Q_\ell \leq 1$.

If the keyword *wl* is set to yes, then the W_ℓ values for each atom will be added to the output array, which are real numbers.

If the keyword *wl/hat* is set to yes, then the $\hat{W}_\ell$ values for each atom will be added to the output array, which are real numbers.

If the keyword *components* is set, then the real and imaginary parts of each component of *normalized* $\hat{Y}_{\ell m}$ will be added to the output array in the following order: $\Re(\hat{Y}_{-m})$, $\Im(\hat{Y}_{-m})$, $\Re(\hat{Y}_{-m+1})$, $\Im(\hat{Y}_{-m+1})$, ..., $\Re(\hat{Y}_{m})$, $\Im(\hat{Y}_{m})$.

In summary, the per-atom array will contain *nlvalues* columns, followed by an additional *nlvalues* columns if *wl* is set to yes, followed by an additional *nlvalues* columns if *wl/hat* is set to yes, followed by an additional 2*(2* *ldegree*+1) columns if the *components* keyword is set.

These values can be accessed by any command that uses per-atom values from a compute as input. See the *Howto output* doc page for an overview of LAMMPS output options.

3.73.5 Restrictions

none

3.73.6 Related commands

compute coord/atom, *compute centro/atom*, *compute hexorder/atom*

3.73.7 Default

The option defaults are *cutoff* = pair style cutoff, *nnn* = 12, *degrees* = 5 4 6 8 10 12 (i.e., Q_4, Q_6, Q_8, Q_{10}, and Q_{12}), *wl* = no, *wl/hat* = no, *components* off, and *chunksize* = 16384

(Steinhardt) P. Steinhardt, D. Nelson, and M. Ronchetti, Phys. Rev. B 28, 784 (1983).

(Mickel) W. Mickel, S. C. Kapfer, G. E. Schroeder-Turkand, K. Mecke, J. Chem. Phys. 138, 044501 (2013).

(tenWolde) P. R. ten Wolde, M. J. Ruiz-Montero, D. Frenkel, J. Chem. Phys. 104, 9932 (1996).

3.74 compute pace command

3.74.1 Syntax

```
compute ID group-ID pace ace_potential_filename ... keyword values ...
```

- ID, group-ID are documented in *compute* command
- pace = style name of this compute command
- ace_potential_filename = file name (in the .yace or .ace format from *pace pair_style*) including ACE hyperparameters, bonds, and generalized coupling coefficients

- keyword = *bikflag* or *dgradflag*

  bikflag value = 0 or 1
    0 = descriptors are summed over atoms of each type
    1 = descriptors are listed separately for each atom
  dgradflag value = 0 or 1
    0 = descriptor gradients are summed over atoms of each type
    1 = descriptor gradients are listed separately for each atom pair

3.74.2 Examples

```
compute pace all pace coupling_coefficients.yace
compute pace all pace coupling_coefficients.yace 0 1
compute pace all pace coupling_coefficients.yace 1 1
```

3.74.3 Description

Added in version 7Feb2024.

This compute calculates a set of quantities related to the atomic cluster expansion (ACE) descriptors of the atoms in a group. ACE descriptors are highly general atomic descriptors, encoding the radial and angular distribution of neighbor atoms, up to arbitrary bond order (rank). The detailed mathematical definition is given in the paper by *(Drautz)*. These descriptors are used in the *pace pair_style*. Quantities obtained from *compute pace* are related to those used in *pace pair_style* to evaluate atomic energies, forces, and stresses for linear ACE models.

For example, the energy for a linear ACE model is calculated as: $E = \sum_i^{N\_atoms} \sum_\nu c_\nu B_{i,\nu}$. The ACE descriptors for atom i $B_{i,\nu}$, and c_ν are linear model parameters. The detailed definition and indexing convention for ACE descriptors is given in *(Drautz)*. In short, body order N, angular character, radial character, and chemical elements in the *N-body* descriptor are encoded by ν. In the *pace pair_style*, the linear model parameters and the ACE descriptors are combined for efficient evaluation of energies and forces. The details and benefits of this efficient implementation are given in *(Lysogorskiy)*, but the combined descriptors and linear model parameters for the purposes of *compute pace* may be expressed in terms of the ACE descriptors mentioned above.

$c_\nu B_{i,\nu} = \sum_{\nu' \in \nu} \left[c_\nu C(\nu')\right] A_{i,\nu'}$

where the bracketed terms on the right-hand side are the combined functions with linear model parameters typically provided in the *<name>.yace* potential file for *pace pair_style*. When these bracketed terms are multiplied by the products of the atomic base from *(Drautz)*, $A_{i,\nu'}$, the ACE descriptors are recovered but they are also scaled by linear model parameters. The generalized coupling coefficients, written in short-hand here as $C(\nu')$, are the generalized Clebsch-Gordan or generalized Wigner symbols. It may be desirable to reverse the combination of these descriptors and the linear model parameters so that the ACE descriptors themselves may be used. The ACE descriptors and their gradients are often used when training ACE models, performing custom data analysis, generalizing ACE model forms, and other tasks that involve direct computation of descriptors. The key utility of *compute pace* is that it can compute the ACE descriptors and gradients so that these tasks can be performed during a LAMMPS simulation or so that LAMMPS can be used as a driver for tasks like ACE model parameterization. To see how this command can be used within a Python workflow to train ACE potentials, see the examples in FitSNAP. Examples on using outputs from this compute to construct general ACE potential forms are demonstrated in *(Goff)*. The various keywords and inputs to *compute pace* determine what ACE descriptors and related quantities are returned in a compute array.

The coefficient file, *<name>.yace*, ultimately defines the number of ACE descriptors to be computed, their maximum body-order, the degree of angular character they have, the degree of radial character they have, the chemical character (which element-element interactions are encoded by descriptors), and other hyper-parameters defined in *(Drautz)*. These may be modeled after the potential files in *pace pair_style*, and have the same format. Details on how to generate the coefficient files to train ACE models may be found in FitSNAP.

The keyword *bikflag* determines whether or not to list the descriptors of each atom separately, or sum them together and list in a single row. If *bikflag* is set to *0* then a single descriptor row is used, which contains the per-atom ACE descriptors $B_{i,v}$ summed over all atoms i to produce B_v. If *bikflag* is set to *1* this is replaced by a separate per-atom ACE descriptor row for each atom. In this case, the entries in the final column for these rows are set to zero.

The keyword *dgradflag* determines whether to sum atom gradients or list them separately. If *dgradflag* is set to 0, the ACE descriptor gradients w.r.t. atom j are summed over all atoms i' of, which may be useful when training linear ACE models on atomic forces. If *dgradflag* is set to 1, gradients are listed separately for each pair of atoms. Each row corresponds to a single term $\frac{\partial B_{i,v}}{\partial r_j^a}$ where r_j^a is the *a-th* position coordinate of the atom with global index j. This also changes the number of columns to be equal to the number of ACE descriptors, with 3 additional columns representing the indices i, j, and a, as explained more in the Output info section below. The option *dgradflag=1* requires that *bikflag=1*.

Note

It is noted here that in contrast to *pace pair_style*, the *.yace* file for *compute pace* typically should not contain linear parameters for an ACE potential. If c_v are included, the value of the descriptor will not be returned in the *compute* array, but instead, the energy contribution from that descriptor will be returned. Do not do this unless it is the desired behavior. *In short, you should not plug in a '.yace' for a pace potential into this compute to evaluate descriptors.*

Note

Generalized Clebsch-Gordan or Generalized Wigner symbols (with appropriate factors) must be used to evaluate ACE descriptors with this compute. There are multiple ways to define the generalized coupling coefficients. Because of this, this compute will not revert your potential file to a coupling coefficient file. Instead this compute allows the user to supply coupling coefficients that follow any convention.

Note

Using *dgradflag* = 1 produces a global array with $N + 3N^2 + 1$ rows which becomes expensive for systems with more than 1000 atoms.

Note

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

3.74.4 Output info

Compute *pace* evaluates a global array. The columns are arranged into *ntypes* blocks, listed in order of atom type *I*. Each block contains one column for each ACE descriptor, the same as for compute *sna/atom*in *compute snap*. A final column contains the corresponding energy, force component on an atom, or virial stress component. The rows of the array appear in the following order:

- 1 row: *pace* average descriptor values for all atoms of type *I*
- 3**n* force rows: quantities, with derivatives w.r.t. x, y, and z coordinate of atom *i* appearing in consecutive rows. The atoms are sorted based on atom ID and run up to the total number of atoms, *n*.
- 6 rows: *virial* quantities summed for all atoms of type *I*

For example, if # $B_{i,v}$ =30 and ntypes=1, the number of columns in the The number of columns in the global array generated by *pace* are 31, and 931, respectively, while the number of rows is 1+3**n*+6, where *n* is the total number of atoms.

If the *bik* keyword is set to 1, the structure of the pace array is expanded. The first *N* rows of the pace array correspond to # $B_{i,v}$ instead of a single row summed over atoms *i*. In this case, the entries in the final column for these rows are set to zero. Also, each row contains only non-zero entries for the columns corresponding to the type of that atom. This is not true in the case of *dgradflag* keyword = 1 (see below).

If the *dgradflag* keyword is set to 1, this changes the structure of the global array completely. Here the per-atom quantities are replaced with rows corresponding to descriptor gradient components on single atoms:

$$\frac{\partial B_{i,v}}{\partial r_j^a}$$

where r_j^a is the *a-th* position coordinate of the atom with global index *j*. The rows are organized in chunks, where each chunk corresponds to an atom with global index *j*. The rows in an atom *j* chunk correspond to atoms with global index *i*. The total number of rows for these descriptor gradients is therefore $3N^2$. The number of columns is equal to the number of ACE descriptors, plus 3 additional left-most columns representing the global atom indices *i*, *j*, and Cartesian direction *a* (0, 1, 2, for x, y, z). The first 3 columns of the first *N* rows belong to the reference potential force components. The remaining K columns contain the $B_{i,v}$ per-atom descriptors corresponding to the non-zero entries obtained when *bikflag* = 1. The first column of the last row, after the first $N+3N^2$ rows, contains the reference potential energy. The virial components are not used with this option. The total number of rows is therefore $N+3N^2+1$ and the number of columns is $K+3$.

These values can be accessed by any command that uses global values from a compute as input. See the *Howto output* doc page for an overview of LAMMPS output options.

3.74.5 Restrictions

These computes are part of the ML-PACE package. They are only enabled if LAMMPS was built with that package. See the *Build package* page for more info.

3.74.6 Related commands

pair_style pace pair_style snap compute snap

3.74.7 Default

The optional keyword defaults are *bikflag* = 0, *dgradflag* = 0

(Drautz) Drautz, Phys Rev B, 99, 014104 (2019).

(Lysogorskiy) Lysogorskiy, van der Oord, Bochkarev, Menon, Rinaldi, Hammerschmidt, Mrovec, Thompson, Csanyi, Ortner, Drautz, npj Comp Mat, 7, 97 (2021).

(Goff) Goff, Zhang, Negre, Rohskopf, Niklasson, Journal of Chemical Theory and Computation 19, no. 13 (2023).

3.75 compute pair command

3.75.1 Syntax

```
compute ID group-ID pair pstyle [nstyle] [evalue]
```

- ID, group-ID are documented in *compute* command
- pair = style name of this compute command
- pstyle = style name of a pair style that calculates additional values
- nsub = *n*-instance of a sub-style, if a pair style is used multiple times in a hybrid style
- *evalue* = *epair* or *evdwl* or *ecoul* or blank (optional)

3.75.2 Examples

```
compute 1 all pair gauss
compute 1 all pair lj/cut/coul/cut ecoul
compute 1 all pair tersoff 2 epair
compute 1 all pair reaxff
```

3.75.3 Description

Define a computation that extracts additional values calculated by a pair style, and makes them accessible for output or further processing by other commands.

Note

The group specified for this command is **ignored**.

The specified *pstyle* must be a pair style used in your simulation either by itself or as a sub-style in a *pair_style hybrid or hybrid/overlay* command. If the sub-style is used more than once, an additional number *nsub* has to be specified in order to choose which instance of the sub-style will be used by the compute. Not specifying the number in this case will cause the compute to fail.

The *evalue* setting is optional. All pair styles tally a potential energy *epair* which may be broken into two parts: *evdwl* and *ecoul* such that *epair* = *evdwl* + *ecoul*. If the pair style calculates Coulombic interactions, their energy will be

tallied in *ecoul*. Everything else (whether it is a Lennard-Jones style van der Waals interaction or not) is tallied in *evdwl*. If *evalue* is blank or specified as *epair*, then *epair* is stored as a global scalar by this compute. This is useful when using *pair_style hybrid* if you want to know the portion of the total energy contributed by one sub-style. If *evalue* is specified as *evdwl* or *ecoul*, then just that portion of the energy is stored as a global scalar.

Note

The energy returned by the *evdwl* keyword does not include tail corrections, even if they are enabled via the *pair_modify* command.

Some pair styles tally additional quantities, e.g. a breakdown of potential energy into 14 components is tallied by the *pair_style reaxff* command. These values (1 or more) are stored as a global vector by this compute. See the page for *individual pair styles* for info on these values.

3.75.4 Output info

This compute calculates a global scalar which is *epair* or *evdwl* or *ecoul*. If the pair style supports it, it also calculates a global vector of length ≥ 1, as determined by the pair style. These values can be used by any command that uses global scalar or vector values from a compute as input. See the *Howto output* doc page for an overview of LAMMPS output options.

The scalar and vector values calculated by this compute are "extensive".

The scalar value will be in energy *units*. The vector values will typically also be in energy *units*, but see the page for the pair style for details.

3.75.5 Restrictions

none

3.75.6 Related commands

compute pe, *compute bond*, *fix pair*

3.75.7 Default

The keyword defaults are *evalue* = *epair*, nsub = 0.

3.76 compute pair/local command

3.76.1 Syntax

```
compute ID group-ID pair/local value1 value2 ... keyword args ...
```

- ID, group-ID are documented in *compute* command
- pair/local = style name of this compute command
- one or more values may be appended

- value = *dist* or *dx* or *dy* or *dz* or *eng* or *force* or *fx* or *fy* or *fz* or *p1* or *p2* or ...

  dist = pairwise distance
  dx,dy,dz = components of pairwise distance
  eng = pairwise energy
  force = pairwise force
  fx,fy,fz = components of pairwise force
  p1, p2, ... = pair style specific quantities for allowed N values

- zero or more keyword/arg pairs may be appended

- keyword = *cutoff*

  cutoff arg = type or radius

3.76.2 Examples

```
compute 1 all pair/local eng
compute 1 all pair/local dist eng force
compute 1 all pair/local dist eng fx fy fz
compute 1 all pair/local dist fx fy fz p1 p2 p3
```

3.76.3 Description

Define a computation that calculates properties of individual pairwise interactions. The number of datums generated, aggregated across all processors, equals the number of pairwise interactions in the system.

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

The value *dist* is the distance between the pair of atoms. The values *dx*, *dy*, and *dz* are the (x, y, z) components of the *distance* between the pair of atoms. This value is always the distance from the atom of higher to the one with the lower atom ID.

The value *eng* is the interaction energy for the pair of atoms.

The value *force* is the force acting between the pair of atoms, which is positive for a repulsive force and negative for an attractive force. The values *fx*, *fy*, and *fz* are the (x, y, z) components of *force* on atom I. For pair styles that apply non-central forces, such as *granular pair styles*, these values only include the (x, y, z) components of the normal force component.

A pair style may define additional pairwise quantities which can be accessed as *p1* to *pN*, where *N* is defined by the pair style. Most pair styles do not define any additional quantities, so $N = 0$. An example of ones that do are the *granular pair styles* which calculate the tangential force between two particles and return its components and magnitude acting on atom *I* for $N \in \{1, 2, 3, 4\}$. See individual pair styles for details.

When using *pN* with pair style *hybrid*, the output will be the Nth quantity from the sub-style that computes the pairwise interaction (based on atom types). If that sub-style does not define a *pN*, the output will be 0.0. The maximum allowed *N* is the maximum number of quantities provided by any sub-style.

When using *pN* with pair style *hybrid/overlay* the quantities from all sub-styles that provide them are concatenated together into one long list. For example, if there are 3 sub-styles and 2 of them have additional output (with 3 and 4 quantities, respectively), then 7 values (*p1* up to *p7*) are defined. The values *p1* to *p3* refer to quantities defined by the first of the two sub-styles. Values *p4* to *p7* refer to quantities from the second of the two sub-styles. If the referenced *pN* is not computed for the specific pairwise interaction (based on atom types), then the output will be 0.0.

The value *dist*, *dx*, *dy* and *dz* will be in distance *units*. The value *eng* will be in energy *units*. The values *force*, *fx*, *fy*, and *fz* will be in force *units*. The values *pN* will be in whatever units the pair style defines.

The optional *cutoff* keyword determines how the force cutoff distance for an interaction is determined. For the default setting of *type*, the pairwise cutoff defined by the *pair_style* command for the types of the two atoms is used. For the *radius* setting, the sum of the radii of the two particles is used as a cutoff. For example, this is appropriate for granular particles which only interact when they are overlapping, as computed by *granular pair styles*. Note that if a granular model defines atom types such that all particles of a specific type are monodisperse (same diameter), then the two settings are effectively identical.

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

Here is an example of how to do this:

```
compute 1 all property/local patom1 patom2
compute 2 all pair/local dist eng force
dump 1 all local 1000 tmp.dump index c_1[1] c_1[2] c_2[1] c_2[2] c_2[3]
```

Note

For pairs, if two atoms I,J are involved in 1–2, 1–3, and 1–4 interactions within the molecular topology, their pairwise interaction may be turned off, and thus they may not appear in the neighbor list, and will not be part of the local data created by this command. More specifically, this will be true of I,J pairs with a weighting factor of 0.0; pairs with a non-zero weighting factor are included. The weighting factors for 1–2, 1–3, and 1–4 pairwise interactions are set by the *special_bonds* command. An exception is if long-range Coulombics are being computed via the *kspace_style* command, then atom pairs with weighting factors of zero are still included in the neighbor list, so that a portion of the long-range interaction contribution can be computed in the pair style. Hence in that case, those atom pairs will be part of the local data created by this command.

3.76.4 Output info

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

The output for *dist* will be in distance *units*. The output for *eng* will be in energy *units*. The output for *force*, *fx*, *fy*, and *fz* will be in force *units*. The output for *pN* will be in whatever units the pair style defines.

3.76.5 Restrictions

none

3.76.6 Related commands

dump local, *compute property/local*

3.76.7 Default

The keyword default is cutoff = type.

3.77 compute pe command

3.77.1 Syntax

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

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

3.77.2 Examples

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

3.77.3 Description

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

The energy is calculated by the various pair, bond, etc. potentials defined for the simulation. If no extra keywords are listed, then the potential energy is the sum of pair, bond, angle, dihedral, improper, *k*-space (long-range), and fix energy (i.e., it is as though all the keywords were listed). If any extra keywords are listed, then only those components are summed to compute the potential energy.

The *k*-space contribution requires 1 extra FFT each timestep the energy is calculated, if using the PPPM solver via the *kspace_style pppm* command. Thus it can increase the cost of the PPPM calculation if it is needed on a large fraction of the simulation timesteps.

Various fixes can contribute to the total potential energy of the system if the *fix* contribution is included. See the doc pages for *individual fixes* for details of which ones compute a potential energy.

> **Note**
>
> The *fix_modify energy yes* command must also be specified if a fix is to contribute potential energy to this command.

A compute of this style with the ID of "thermo_pe" is created when LAMMPS starts up, as if this command were in the input script:

```
compute thermo_pe all pe
```

See the "thermo_style" command for more details.

3.77.4 Output info

This compute calculates a global scalar (the potential energy). This value can be used by any command that uses a global scalar value from a compute as input. See the *Howto output* doc page for an overview of LAMMPS output options.

The scalar value calculated by this compute is "extensive". The scalar value will be in energy *units*.

3.77.5 Restrictions

none

3.77.6 Related commands

compute pe/atom

3.77.7 Default

none

3.78 compute pe/atom command

3.78.1 Syntax

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

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

3.78.2 Examples

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

3.78.3 Description

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

The per-atom energy is calculated by the various pair, bond, etc potentials defined for the simulation. If no extra keywords are listed, then the potential energy is the sum of pair, bond, angle, dihedral, improper, *k*-space (long-range), and fix energy (i.e., it is as though all the keywords were listed). If any extra keywords are listed, then only those components are summed to compute the potential energy.

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

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

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

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

Various fixes can contribute to the per-atom potential energy of the system if the *fix* contribution is included. See the doc pages for *individual fixes* for details of which ones compute a per-atom potential energy.

Note

The *fix_modify energy yes* command must also be specified if a fix is to contribute per-atom potential energy to this command.

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

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

Note

The per-atom energy does not include any Lennard-Jones tail corrections to the energy added by the *pair_modify tail yes* command, since those are contributions to the global system energy.

3.78.4 Output info

This compute calculates a per-atom vector, which can be accessed by any command that uses per-atom values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

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

3.78.5 Restrictions

3.78.6 Related commands

compute pe, *compute stress/atom*

3.78.7 Default

none

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

3.79 compute plasticity/atom command

3.79.1 Syntax

```
compute ID group-ID plasticity/atom
```

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

3.79.2 Examples

```
compute 1 all plasticity/atom
```

3.79.3 Description

Define a computation that calculates the per-atom plasticity for each atom in a group. This is a quantity relevant for *Peridynamics models*. See this document for an overview of LAMMPS commands for Peridynamics modeling.

The plasticity for a Peridynamic particle is the so-called consistency parameter (λ). For elastic deformation, $\lambda = 0$, otherwise $\lambda > 0$ for plastic deformation. For details, see *(Mitchell)* and the PDF doc included in the LAMMPS distribution in doc/PDF/PDLammps_EPS.pdf.

This command can be invoked for one of the Peridynamic *pair styles*: peri/eps.

The plasticity value will be 0.0 for atoms not in the specified compute group.

3.79.4 Output info

This compute calculates a per-atom vector, which can be accessed by any command that uses per-atom values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

The per-atom vector values are unitless numbers $\lambda \geq 0.0$.

3.79.5 Restrictions

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

3.79.6 Related commands

compute damage/atom, *compute dilatation/atom*

3.79.7 Default

none

(Mitchell) Mitchell, "A non-local, ordinary-state-based viscoelasticity model for peridynamics", Sandia National Lab Report, 8064:1-28 (2011).

3.80 compute pod/atom command

3.81 compute podd/atom command

3.82 compute pod/local command

3.83 compute pod/global command

3.83.1 Syntax

```
compute ID group-ID pod/atom param.pod coefficients.pod
compute ID group-ID podd/atom param.pod coefficients.pod
compute ID group-ID pod/local param.pod coefficients.pod
compute ID group-ID pod/global param.pod coefficients.pod
```

- ID, group-ID are documented in *compute* command
- pod/atom = style name of this compute command
- param.pod = the parameter file specifies parameters of the POD descriptors
- coefficients.pod = the coefficient file specifies coefficients of the POD potential

3.83.2 Examples

```
compute d all pod/atom Ta_param.pod
compute dd all podd/atom Ta_param.pod
compute ldd all pod/local Ta_param.pod
compute gdd all podd/global Ta_param.pod
compute d all pod/atom Ta_param.pod Ta_coefficients.pod
compute dd all podd/atom Ta_param.pod Ta_coefficients.pod
compute ldd all pod/local Ta_param.pod Ta_coefficients.pod
compute gdd all podd/global Ta_param.pod Ta_coefficients.pod
```

3.83.3 Description

Added in version 27June2024.

Define a computation that calculates a set of quantities related to the POD descriptors of the atoms in a group. These computes are used primarily for calculating the dependence of energy and force components on the linear coefficients in the *pod pair_style*, which is useful when training a POD potential to match target data. POD descriptors of an atom are characterized by the radial and angular distribution of neighbor atoms. The detailed mathematical definition is given in the papers by *(Nguyen and Rohskopf)*, *(Nguyen2023)*, *(Nguyen2024)*, and *(Nguyen and Sema)*.

Compute *pod/atom* calculates the per-atom POD descriptors.

Compute *podd/atom* calculates derivatives of the per-atom POD descriptors with respect to atom positions.

Compute *pod/local* calculates the per-atom POD descriptors and their derivatives with respect to atom positions.

Compute *pod/global* calculates the global POD descriptors and their derivatives with respect to atom positions.

Examples how to use Compute POD commands are found in the directory examples/PACKAGES/pod.

Warning

All of these compute styles produce *very* large per-atom output arrays that scale with the total number of atoms in the system. This will result in *very* large memory consumption for systems with a large number of atoms.

3.83.4 Output info

Compute *pod/atom* produces an 2D array of size $N \times M$, where N is the number of atoms and M is the number of descriptors. Each column corresponds to a particular POD descriptor.

Compute *podd/atom* produces an 2D array of size $N \times (M * 3N)$. Each column corresponds to a particular derivative of a POD descriptor.

Compute *pod/local* produces an 2D array of size $(1 + 3N) \times (M * N)$. The first row contains the per-atom descriptors, and the last 3N rows contain the derivatives of the per-atom descriptors with respect to atom positions.

Compute *pod/global* produces an 2D array of size $(1 + 3N) \times (M)$. The first row contains the global descriptors, and the last 3N rows contain the derivatives of the global descriptors with respect to atom positions.

3.83.5 Restrictions

These computes are part of the ML-POD package. They are only enabled if LAMMPS was built with that package. See the *Build package* page for more info.

3.83.6 Related commands

fitpod, *pair_style pod*

3.83.7 Default

none

(Nguyen and Rohskopf) Nguyen and Rohskopf, Journal of Computational Physics, 480, 112030, (2023).

(Nguyen2023) Nguyen, Physical Review B, 107(14), 144103, (2023).

(Nguyen2024) Nguyen, Journal of Computational Physics, 113102, (2024).

(Nguyen and Sema) Nguyen and Sema, https://arxiv.org/abs/2405.00306, (2024).

3.84 compute pressure command

3.84.1 Syntax

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

- ID, group-ID are documented in *compute* command
- pressure = style name of this compute command
- temp-ID = ID of compute that calculates temperature, can be NULL if not needed
- zero or more keywords may be appended
- keyword = *ke* or *pair* or *bond* or *angle* or *dihedral* or *improper* or *kspace* or *fix* or *virial* or *pair/hybrid*

3.84.2 Examples

```
compute 1 all pressure thermo_temp
compute 1 all pressure NULL pair bond
compute 1 all pressure NULL pair/hybrid lj/cut
```

3.84.3 Description

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

The pressure is computed by the formula

$$P = \frac{Nk_BT}{V} + \frac{1}{Vd}\sum_{i=1}^{N'} \vec{r}_i \cdot \vec{f}_i$$

where N is the number of atoms in the system (see discussion of DOF below), k_B is the Boltzmann constant, T is the temperature, d is the dimensionality of the system (2 for 2d, 3 for 3d), and V is the system volume (or area in 2d). The second term is the virial, equal to $-dU/dV$, computed for all pairwise as well as 2-body, 3-body, 4-body, many-body, and long-range interactions, where $\vec{r}_i$ and $\vec{f}_i$ are the position and force vector of atom i, and the dot indicates the dot product (scalar product). This is computed in parallel for each subdomain and then summed over all parallel processes. Thus N' necessarily includes atoms from neighboring subdomains (so-called ghost atoms) and the position and force vectors of ghost atoms are thus included in the summation. Only when running in serial and without periodic boundary conditions is $N' = N$ the number of atoms in the system. *Fixes* that impose constraints (e.g., the *fix shake* command) may also contribute to the virial term.

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

$$P_{IJ} = \frac{1}{V}\sum_{k=1}^{N} m_k v_{k_I} v_{k_J} + \frac{1}{V}\sum_{k=1}^{N'} r_{k_I} f_{k_J}.$$

If no extra keywords are listed, the entire equations above are calculated. This includes a kinetic energy (temperature) term and the virial as the sum of pair, bond, angle, dihedral, improper, kspace (long-range), and fix contributions to the force on each atom. If any extra keywords are listed, then only those components are summed to compute temperature or ke and/or the virial. The *virial* keyword means include all terms except the kinetic energy *ke*.

The *pair/hybrid* keyword means to only include contribution from a sub-style in a *hybrid* or *hybrid/overlay* pair style.

Details of how LAMMPS computes the virial efficiently for the entire system, including for many-body potentials and accounting for the effects of periodic boundary conditions are discussed in *(Thompson)*.

The temperature and kinetic energy tensor are not calculated by this compute, but rather by the temperature compute specified with the command. See the doc pages for individual compute temp variants for an explanation of how they calculate temperature and a symmetric tensor (6-element vector) whose components are twice that of the traditional KE tensor. That tensor is what appears in the pressure tensor formula above.

If the kinetic energy is not included in the pressure, than the temperature compute is not used and can be specified as NULL. Normally the temperature compute used by compute pressure should calculate the temperature of all atoms for consistency with the virial term, but any compute style that calculates temperature can be used (e.g., one that excludes frozen atoms or other degrees of freedom).

Note that if desired the specified temperature compute can be one that subtracts off a bias to calculate a temperature using only the thermal velocity of the atoms (e.g., by subtracting a background streaming velocity). See the doc pages for individual *compute commands* to determine which ones include a bias.

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

A compute of this style with the ID of thermo_press is created when LAMMPS starts up, as if this command were in the input script:

```
compute thermo_press all pressure thermo_temp
```

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

Styles with a *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.

3.84.4 Output info

This compute calculates a global scalar (the pressure) and a global vector of length 6 (pressure tensor), which can be accessed by indices 1–6. These values can be used by any command that uses global scalar or vector values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

The ordering of values in the symmetric pressure tensor is as follows: p_{xx}, p_{yy}, p_{zz}, p_{xy}, p_{xz}, p_{yz}.

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

3.84.5 Restrictions

none

3.84.6 Related commands

compute temp, *compute stress/atom*, *thermo_style*, *fix numdiff/virial*,

3.84.7 Default

By default the compute includes contributions from the keywords: ke pair bond angle dihedral improper kspace fix

---

(Thompson) Thompson, Plimpton, Mattson, J Chem Phys, 131, 154107 (2009).

3.85 compute pressure/alchemy command

3.85.1 Syntax

```
compute ID group-ID pressure/alchemy fix-ID
```

- ID, group-ID are documented in *compute* command
- pressure/alchemy = style name of this compute command
- fix-ID = ID of *fix alchemy* command

3.85.2 Examples

```
fix trans all alchemy
compute mixed all pressure/alchemy trans
thermo_modify press mixed
```

3.85.3 Description

Added in version 28Mar2023.

Define a compute style that makes the "mixed" system pressure available for a system that uses the *fix alchemy* command to transform one topology to another. This can be used in combination with either *thermo_modify press* or *fix_modify press* to output and access a pressure consistent with the simulated combined two topology system.

The actual pressure is determined with *compute pressure* commands that are internally used by *fix alchemy* for each topology individually and then combined. This command just extracts the information from the fix.

The examples/PACKAGES/alchemy folder contains an example input for this command.

3.85.4 Output info

This compute calculates a global scalar (the pressure) and a global vector of length 6 (the pressure tensor), which can be accessed by indices 1–6. These values can be used by any command that uses global scalar or vector values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

The ordering of values in the symmetric pressure tensor is as follows: p_{xx}, p_{yy}, p_{zz}, p_{xy}, p_{xz}, p_{yz}.

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

3.85.5 Restrictions

This compute 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.

3.85.6 Related commands

fix alchemy, *compute pressure*, *thermo_modify*, *fix_modify*

3.85.7 Default

none

3.86 compute pressure/uef command

3.86.1 Syntax

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

- ID, group-ID are documented in *compute* command
- pressure/uef = style name of this compute command
- temp-ID = ID of compute that calculates temperature, can be NULL if not needed
- zero or more keywords may be appended
- keyword = *ke* or *pair* or *bond* or *angle* or *dihedral* or *improper* or *kspace* or *fix* or *virial*

3.86.2 Examples

```
compute 1 all pressure/uef my_temp_uef
compute 2 all pressure/uef my_temp_uef virial
```

3.86.3 Description

This command is used to compute the pressure tensor in the reference frame of the applied flow field when *fix nvt/uef* or *fix npt/uef* is used. It is not necessary to use this command to compute the scalar value of the pressure. A *compute pressure* may be used for that purpose.

The keywords and output information are documented in *compute_pressure*.

3.86.4 Restrictions

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

This command can only be used when *fix nvt/uef* or *fix npt/uef* is active.

The kinetic contribution to the pressure tensor will be accurate only when the compute specified by *temp-ID* is a *compute temp/uef*.

3.86.5 Related commands

compute pressure, *fix nvt/uef*, *compute temp/uef*

3.86.6 Default

none

3.87 compute property/atom command

3.87.1 Syntax

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

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

```
possible attributes = id, mol, proc, type, mass,
                      x, y, z, xs, ys, zs, xu, yu, zu, ix, iy, iz,
                      vx, vy, vz, fx, fy, fz,
                      q, mux, muy, muz, mu,
                      spx, spy, spz, sp, fmx, fmy, fmz,
                      nbonds,
                      radius, diameter, omegax, omegay, omegaz,
                      temperature, heatflow,
                      angmomx, angmomy, angmomz,
                      shapex, shapey, shapez,
                      quatw, quati, quatj, quatk, tqx, tqy, tqz,
                      end1x, end1y, end1z, end2x, end2y, end2z,
                      corner1x, corner1y, corner1z,
                      corner2x, corner2y, corner2z,
                      corner3x, corner3y, corner3z,
                      i_name, d_name, i2_name[I], d2_name[I],
                      vfrac, s0, espin, eradius, ervel, erforce,
                      rho, drho, e, de, cv, buckling,
```

id = atom ID
mol = molecule ID
proc = ID of processor that owns atom

type = atom type
mass = atom mass
x,y,z = unscaled atom coordinates
xs,ys,zs = scaled atom coordinates
xu,yu,zu = unwrapped atom coordinates
ix,iy,iz = box image that the atom is in
vx,vy,vz = atom velocities
fx,fy,fz = forces on atoms
q = atom charge
mux,muy,muz = orientation of dipole moment of atom
mu = magnitude of dipole moment of atom
spx, spy, spz = direction of the atomic magnetic spin
sp = magintude of atomic magnetic spin moment
fmx, fmy, fmz = magnetic force
nbonds = number of bonds assigned to an atom
radius,diameter = radius,diameter of spherical particle
omegax,omegay,omegaz = angular velocity of spherical particle
temperature = internal temperature of spherical particle
heatflow = internal heat flow of spherical particle
angmomx,angmomy,angmomz = angular momentum of aspherical particle
shapex,shapey,shapez = 3 diameters of aspherical particle
quatw,quati,quatj,quatk = quaternion components for aspherical or body particles
tqx,tqy,tqz = torque on finite-size particles
end12x, end12y, end12z = end points of line segment
corner123x, corner123y, corner123z = corner points of triangle
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

PERI package per-atom properties:
vfrac = volume fraction
s0 = max stretch of any bond a particle is part of

EFF and AWPMD package per-atom properties:
espin = electron spin
eradius = electron radius
ervel = electron radial velocity
erforce = electron radial force

SPH package per-atom properties:
rho = density of SPH particles
drho = change in density
e = energy
de = change in thermal energy
cv = heat capacity

3.87.2 Examples

```
compute 1 all property/atom xs vx fx mux
compute 2 all property/atom type
compute 1 all property/atom ix iy iz
compute 3 all property/atom sp spx spy spz
compute 1 all property/atom i_myFlag d_Sxyz[1] d_Sxyz[3]
```

3.87.3 Description

Define a computation that simply stores atom attributes for each atom in the group. This is useful so that the values can be used by other *output commands* that take computes as inputs. See for example, the *compute reduce*, *fix ave/atom*, *fix ave/histo*, *fix ave/chunk*, and *atom-style variable* commands.

The list of possible attributes is essentially the same as that used by the *dump custom* command, which describes their meaning, with some additional quantities that are only defined for certain *atom styles*. The goal of this augmented list gives an input script access to any per-atom quantity stored by LAMMPS.

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

Attributes *i_name*, *d_name*, *i2_name*, *d2_name* refer to custom per-atom integer and floating-point vectors or arrays that have been added via the *fix property/atom* command. When that command is used specific names are given to each attribute which are the "name" portion of these attributes. For arrays *i2_name* and *d2_name*, the column of the array must also be included following the name in brackets (e.g., d2_xyz[2] or i2_mySpin[3]).

The additional quantities only accessible via this command, and not directly via the *dump custom* command, are as follows.

Nbonds is available for all molecular atom styles and refers to the number of explicit bonds assigned to an atom. Note that if the *newton bond* command is set to *on*, which is the default, then every bond in the system is assigned to only one of the two atoms in the bond. Thus a bond between atoms *I* and *J* may be tallied for either atom *I* or atom *J*. If *newton bond off* is set, it will be tallied with both atom *I* and atom *J*.

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

The quantities *quatw*, *quati*, *quatj*, and *quatk* are defined for ellipsoidal particles and body particles and store the 4-vector quaternion representing the orientation of each particle. See the *set* command for an explanation of the quaternion vector.

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

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

In addition, the various per-atom quantities listed above for specific packages are only accessible by this command.

Changed in version 15Sep2022: The *espin* property was previously called *spin*.

3.87.4 Output info

This compute calculates a per-atom vector or per-atom array depending on the number of input values. If a single input is specified, a per-atom vector is produced. If two or more inputs are specified, a per-atom array is produced where the number of columns = the number of inputs. The vector or array can be accessed by any command that uses per-atom values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

The vector or array values will be in whatever *units* the corresponding attribute is in (e.g., velocity units for *vx*, charge units for *q*).

For the spin quantities, *sp* is in the units of the Bohr magneton; *spx*, *spy*, and *spz* are unitless quantities; and *fmx*, *fmy*, and *fmz* are given in rad/THz.

3.87.5 Restrictions

none

3.87.6 Related commands

dump custom, *compute reduce*, *fix ave/atom*, *fix ave/chunk*, *fix property/atom*

3.87.7 Default

none

3.88 compute property/chunk command

3.88.1 Syntax

```
compute ID group-ID property/chunk chunkID input1 input2 ...
```

- ID, group-ID are documented in *compute* command
- property/chunk = style name of this compute command
- chunkID = ID of *compute chunk/atom* command that defines the chunks
- input1,etc = one or more attributes

  attributes = count, id, coord1, coord2, coord3
    count = # of atoms in chunk
    id = original chunk IDs before compression by compute chunk/atom
    coord123 = coordinates for spatial bins calculated by compute chunk/atom

3.88.2 Examples

```
compute 1 all property/chunk bin2d id count
compute 1 all property/chunk myChunks id coord1
```

3.88.3 Description

Define a computation that stores the specified attributes of chunks of atoms.

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* and *Howto chunk* doc pages for details of how chunks can be defined and examples of how they can be used to measure properties of a system.

This compute calculates and stores the specified attributes of chunks as global data so they can be accessed by other *output commands* and used in conjunction with other commands that generate per-chunk data, such as *compute com/chunk* or *compute msd/chunk*.

Note that only atoms in the specified group contribute to the calculation of the *count* attribute. The *compute chunk/atom* command defines its own group; atoms will have a chunk ID = 0 if they are not in that group, signifying they are not assigned to a chunk, and will thus also not contribute to this calculation. You can specify the "all" group for this command if you simply want to include atoms with non-zero chunk IDs.

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

The *id* attribute stores the original chunk ID for each chunk. It can only be used if the *compress* keyword was set to *yes* for the *compute chunk/atom* command referenced by chunkID. 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 (molecule ID in this case) of 415. The *id* attribute will then be 415 for the third chunk.

The *coordN* attributes can only be used if a *binning* style was used

in the *compute chunk/atom* command referenced by chunkID. For *bin/1d*, *bin/2d*, and *bin/3d* styles the attribute is the center point of the bin in the corresponding dimension. Style *bin/1d* only defines a *coord1* attribute. Style *bin/2d* adds a *coord2* attribute. Style *bin/3d* adds a *coord3* attribute.

Note that if the value of the *units* keyword used in the *compute chunk/atom command* is *box* or *lattice*, the *coordN* attributes will be in distance *units*. If the value of the *units* keyword is *reduced*, the *coordN* attributes will be in unitless reduced units (0-1).

The simplest way to output the results of the compute property/chunk calculation to a file is to use the *fix ave/time* command, for example:

```
compute cc1 all chunk/atom molecule
compute myChunk1 all property/chunk cc1 count
compute myChunk2 all com/chunk cc1
fix 1 all ave/time 100 1 100 c_myChunk1 c_myChunk2[*] file tmp.out mode vector
```

3.88.4 Output info

This compute calculates a global vector or global array depending on the number of input values. The length of the vector or number of rows in the array is the number of chunks.

This compute calculates a global vector or global array where the number of rows = the number of chunks *Nchunk* as calculated by the specified *compute chunk/atom* command. If a single input is specified, a global vector is produced. If two or more inputs are specified, a global array is produced where the number of columns = the number of inputs. The vector or array can be accessed by any command that uses global values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

The vector or array values are "intensive". The values will be unitless or in the units discussed above.

3.88.5 Restrictions

none

3.88.6 Related commands

fix ave/chunk

3.88.7 Default

none

3.89 compute property/grid command

3.89.1 Syntax

```
compute ID group-ID property/grid Nx Ny Nz input1 input2 ...
```

- ID, group-ID are documented in *compute* command
- property/grid = style name of this compute command
- Nx, Ny, Nz = grid size in each dimension
- input1,etc = one or more attributes

```
attributes = id, ix, iy, iz, x, y, z, xs, ys, zs, xc, yc, zc, xsc, ysc, zsc
  id = ID of grid cell, x fastest, y next, z slowest
  proc = processor ID (0 to Nprocs-1) which owns the grid cell
  ix,iy,iz = grid indices in each dimension (1 to N inclusive)
  x,y,z = coords of lower left corner of grid cell
  xs,ys,zs = scaled coords of lower left corner of grid cell (0.0 to 1.0)
  xc,yc,zc = coords of center point of grid cell
  xsc,ysc,zsc = scaled coords of center point of grid cell (0.0 to 1.0)
```

3.89.2 Examples

```
compute 1 all property/grid 10 10 20 id ix iy iz
compute 1 all property/grid 100 100 1 id xc yc zc
```

3.89.3 Description

Define a computation that stores the specified attributes of a distributed grid. In LAMMPS, distributed grids are regular 2d or 3d grids which overlay a 2d or 3d simulation domain. Each processor owns the grid cells whose center points lie within its subdomain. See the *Howto grid* doc page for details of how distributed grids can be defined by various commands and referenced.

This compute stores the specified attributes of grids as per-grid data so they can be accessed by other *output commands* such as *dump grid*.

Nx, *Ny*, and *Nz* define the size of the grid. For a 2d simulation *Nz* must be 1. When this compute is used by *dump grid*, to output per-grid values from other computes of fixes, the grid size specified for this command must be consistent with the grid sizes used by the other commands.

The *id* attribute is the grid ID for each grid cell. For a global grid of size Nx by Ny by Nz (in 3d simulations) the grid IDs range from 1 to Nx*Ny*Nz. They are ordered with the X index of the 3d grid varying fastest, then Y, then Z slowest. For 2d grids (in 2d simulations), the grid IDs range from 1 to Nx*Ny, with X varying fastest and Y slowest.

Added in version 21Nov2023.

The *proc* attribute is the ID of the processor which owns the grid cell. Processor IDs range from 0 to Nprocs - 1, where Nprocs is the number of processors the simulation is running on. Each grid cell is owned by a single processor.

The *ix*, *iy*, *iz* attributes are the indices of a grid cell in each dimension. They range from 1 to Nx inclusive in the X dimension, and similar for Y and Z.

The *x*, *y*, *z* attributes are the coordinates of the lower left corner point of each grid cell.

The *xs*, *ys*, *zs* attributes are also coordinates of the lower left corner point of each grid cell, except in scaled coordinates, where the lower-left corner of the entire simulation box is (0,0,0) and the upper right corner is (1,1,1).

The *xc*, *yc*, *zc* attributes are the coordinates of the center point of each grid cell.

The *xsc*, *ysc*, *zsc* attributes are also coordinates of the center point each grid cell, except in scaled coordinates, where the lower-left corner of the entire simulation box is (0,0,0) and the upper right corner is (1,1,1).

For *triclinic simulation boxes*, the grid point coordinates for (x,y,z) and (xc,yc,zc) will reflect the triclinic geometry. For (xs,yz,zs) and (xsc,ysc,zsc), the coordinates are the same for orthogonal versus triclinic boxes.

3.89.4 Output info

This compute calculates a per-grid vector or array depending on the number of input values. The length of the vector or number of array rows (distributed across all processors) is Nx * Ny * Nz. For access by other commands, the name of the single grid produced by this command is “grid”. The name of its per-grid data is “data”.

The (x,y,z) and (xc,yc,zc) coordinates are in distance *units*.

3.89.5 Restrictions

For 2d simulations, the attributes which refer to the Z dimension cannot be used.

3.89.6 Related commands

dump grid

3.89.7 Default

none

3.90 compute property/local command

3.90.1 Syntax

```
compute ID group-ID property/local attribute1 attribute2 ... keyword args ...
```

- ID, group-ID are documented in *compute* command
- property/local = style name of this compute command
- one or more attributes of the same type (neighbor, pair, bond, angle, dihedral, or improper) may be appended

```
possible attributes = natom1, natom2, ntype1, ntype2,
                      patom1, patom2, ptype1, ptype2,
                      batom1, batom2, btype,
                      aatom1, aatom2, aatom3, atype,
                      datom1, datom2, datom3, datom4, dtype,
                      iatom1, iatom2, iatom3, iatom4, itype
```

  - Neighbor attributes

```
natom1, natom2 = store IDs of two atoms in each pair (within neighbor cutoff)
ntype1, ntype2 = store types of two atoms in each pair (within neighbor cutoff)
```

  - Pair attributes

```
patom1, patom2 = store IDs of two atoms in each pair (within force cutoff)
ptype1, ptype2 = store types of two atoms in each pair (within force cutoff)
```

  - Bond attributes

```
batom1, batom2 = store IDs of two atoms in each bond
btype = store bond type of each bond
```

  - Angle attributes

```
aatom1, aatom2, aatom3 = store IDs of three atoms in each angle
atype = store angle type of each angle
```

  - Dihedral attributes

```
datom1, datom2, datom3, datom4 = store IDs of 4 atoms in each dihedral
dtype = store dihedral type of each dihedral
```

  - Improper attributes

```
iatom1, iatom2, iatom3, iatom4 = store IDs of 4 atoms in each improper
itype = store improper type of each improper
```

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

  cutoff arg = type or radius

3.90.2 Examples

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

3.90.3 Description

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

If multiple attributes are specified then they must all generate the same amount of information, so that the resulting local array has the same number of rows for each column. This means that only bond attributes can be specified together, or angle attributes, etc. Bond and angle attributes cannot be mixed in the same compute property/local command.

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

The optional *cutoff* keyword determines how the force cutoff distance for an interaction is determined for the *patom1* and *patom2* attributes. For the default setting of *type*, the pairwise cutoff defined by the *pair_style* command for the types of the two atoms is used. For the *radius* setting, the sum of the radii of the two particles is used as a cutoff. For example, this is appropriate for granular particles which only interact when they are overlapping, as computed by *granular pair styles*. Note that if a granular model defines atom types such that all particles of a specific type are monodisperse (same diameter), then the two settings are effectively identical.

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

For bonds and angles, a bonds/angles that have been broken by setting their bond/angle type to 0 will not be included. Bonds/angles that have been turned off (see the *fix shake* or *delete_bonds* commands) by setting their bond/angle type negative are written into the file. This is consistent with the *compute bond/local* and *compute angle/local* commands

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

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

Note

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

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

The attributes that start with "a", "d", and "i" refer to similar values for *angles*, *dihedrals*, and *impropers*.

3.90.4 Output info

This compute calculates a local vector or local array depending on the number of input values. The length of the vector or number of rows in the array is the number of bonds, angles, etc. If a single input is specified, a local vector is produced. If two or more inputs are specified, a local array is produced where the number of columns = the number of inputs. The vector or array can be accessed by any command that uses local values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

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

3.90.5 Restrictions

none

3.90.6 Related commands

dump local, *compute reduce*

3.90.7 Default

The keyword default is cutoff = type.

3.91 compute ptm/atom command

3.91.1 Syntax

```
compute ID group-ID ptm/atom structures threshold group2-ID
```

- ID, group-ID are documented in *compute* command
- ptm/atom = style name of this compute command
- structures = *default* or *all* or any hyphen-separated combination of *fcc*, *hcp*, *bcc*, *ico*, *sc*, *dcub*, *dhex*, or *graphene* = structure types to search for
- threshold = lattice distortion threshold (RMSD)
- group2-ID determines which group is used for neighbor selection (optional, default "all")

3.91.2 Examples

```
compute 1 all ptm/atom default 0.1 all
compute 1 all ptm/atom fcc-hcp-dcub-dhex 0.15 all
compute 1 all ptm/atom all 0
```

3.91.3 Description

Define a computation that determines the local lattice structure around an atom using the PTM (Polyhedral Template Matching) method. The PTM method is described in *(Larsen)*.

Currently, there are seven lattice structures PTM recognizes:

- fcc = 1
- hcp = 2
- bcc = 3
- ico (icosahedral) = 4
- sc (simple cubic) = 5
- dcub (diamond cubic) = 6
- dhex (diamond hexagonal) = 7
- graphene = 8

The value of the PTM structure will be 0 for unknown types and -1 for atoms not in the specified compute group. The choice of structures to search for can be specified using the "structures" argument, which is a hyphen-separated list of structure keywords. Two convenient pre-set options are provided:

- default: fcc-hcp-bcc-ico
- all: fcc-hcp-bcc-ico-sc-dcub-dhex-graphene

The 'default' setting detects the same structures as the Common Neighbor Analysis method. The 'all' setting searches for all structure types. A performance penalty is incurred for the diamond and graphene structures, so it is not recommended to use this option if it is known that the simulation does not contain these structures.

PTM identifies structures using two steps. First, a graph isomorphism test is used to identify potential structure matches. Next, the deviation is computed between the local structure (in the simulation) and a template of the ideal lattice structure. The deviation is calculated as:

$$\mathrm{RMSD}(\mathbf{u},\mathbf{v}) = \min_{s,\mathbf{Q}} \sqrt{\frac{1}{N}\sum_{i=1}^{N} \left\| s[\vec{u}_i - \bar{\mathbf{u}}] - \mathbf{Q}\cdot\vec{v}_i \right\|^2}$$

Here, $\vec{u}$ and $\vec{v}$ contain the coordinates of the local and ideal structures respectively, s is a scale factor, and $\mathbf{Q}$ is a rotation. The best match is identified by the lowest RMSD value, using the optimal scaling, rotation, and correspondence between the points.

The *threshold* keyword sets an upper limit on the maximum permitted deviation before a local structure is identified as disordered. Typical values are in the range 0.1–0.15, but larger values may be desirable at higher temperatures. A value of 0 is equivalent to infinity and can be used if no threshold is desired.

The neighbor list needed to compute this quantity is constructed each time the calculation is performed (e.g., each time a snapshot of atoms is dumped). Thus it can be inefficient to compute/dump this quantity too frequently or to have multiple compute/dump commands, each with a *ptm/atom* style. By default the compute processes **all** neighbors unless the optional *group2-ID* argument is given, then only members of that group are considered as neighbors.

3.91.4 Output info

This compute calculates a per-atom array, which can be accessed by any command that uses per-atom values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

Results are stored in the per-atom array in the following order:

- type
- rmsd
- interatomic distance
- qw
- qx
- qy
- qz

The type is a number from -1 to 8. The rmsd is a positive real number. The interatomic distance is computed from the scale factor in the RMSD equation. The (qw, qx, qy, qz) parameters represent the orientation of the local structure in quaternion form. The reference coordinates for each template (from which the orientation is determined) can be found in the *ptm_constants.h* file in the PTM source directory. For atoms that are not within the compute group-ID, all values are set to zero.

3.91.5 Restrictions

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

3.91.6 Related commands

compute centro/atom compute cna/atom

3.91.7 Default

none

(Larsen) Larsen, Schmidt, Schiotz, Modelling Simul Mater Sci Eng, 24, 055007 (2016).

3.92 compute rattlers/atom command

3.92.1 Syntax

```
compute ID group-ID rattlers/atom cutoff zmin ntries
```

- ID, group-ID are documented in *compute* command
- rattlers/atom = style name of this compute command
- cutoff = *type* or *radius*

  type = cutoffs determined based on atom types
  radius = cutoffs determined based on atom diameters (atom style sphere)
- zmin = minimum coordination for a non-rattler atom
- ntries = maximum number of iterations to remove rattlers

3.92.2 Examples

```
compute 1 all rattlers/atom type 4 10
```

3.92.3 Description

Added in version 7Feb2024.

Define a compute that identifies rattlers in a system. Rattlers are often identified in granular or glassy packings as under-coordinated atoms that do not have the required number of contacts to constrain their translational degrees of freedom. Such atoms are not considered rigid and can often freely rattle around in the system. This compute identifies rattlers which can be helpful for excluding them from analysis or providing extra damping forces to accelerate relaxation processes.

Rattlers are identified using an interactive approach. The coordination number of all atoms is first calculated. The *type* and *radius* settings are used to select whether interaction cutoffs are determined by atom types or by the sum of atomic radii (atom style sphere), respectively. Rattlers are then identified as atoms with a coordination number less than *zmin* and are removed from consideration. Atomic coordination numbers are then recalculated, excluding previously identified rattlers, to identify a new set of rattlers. This process is iterated up to a maximum of *ntries* or until no new rattlers are identified and the remaining atoms form a stable network of contacts.

In dense homogeneous systems where the average atom coordination number is expected to be larger than *zmin*, this process usually only takes a few iterations and a value of *ntries* around ten may be sufficient. In systems with significant heterogeneity or average coordination numbers less than *zmin*, an appropriate value of *ntries* depends heavily on the specific system. For instance, a linear chain of N rattler atoms with a *zmin* of 2 would take N/2 iterations to identify that all the atoms are rattlers.

3.92.4 Output info

This compute calculates a per-atom vector and a global scalar. The vector designates which atoms are rattlers, indicated by a value 1. Non-rattlers have a value of 0. The global scalar returns the total number of rattlers in the system. See the *Howto output* page for an overview of LAMMPS output options.

3.92.5 Restrictions

This compute is part of the EXTRA-COMPUTE package. It is only enabled if LAMMPS was built with that package. See the *Build package* page for more info.

The *radius* cutoff option requires that atoms store a radius as defined by the *atom_style sphere* or similar commands.

3.92.6 Related commands

compute coord/atom compute contact/atom

3.92.7 Default

none

3.93 compute rdf command

3.93.1 Syntax

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

- ID, group-ID are documented in *compute* command
- rdf = style name of this compute command
- Nbin = number of RDF bins
- itypeN = central atom type for Nth RDF histogram (integer, type label, or asterisk form)
- jtypeN = distribution atom type for Nth RDF histogram (integer, type label, or asterisk form)
- zero or more keyword/value pairs may be appended
- keyword = *cutoff*

  cutoff value = Rcut
    Rcut = cutoff distance for RDF computation (distance units)

3.93.2 Examples

```
compute 1 all rdf 100
compute 1 all rdf 100 1 1
compute 1 all rdf 100 * 3 cutoff 5.0
compute 1 fluid rdf 500 1 1 1 2 2 1 2 2
compute 1 fluid rdf 500 1*3 2 5 *10 cutoff 3.5
```

3.93.3 Description

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

Note

If you have a bonded system, then the settings of *special_bonds* command can remove pairwise interactions between atoms in the same bond, angle, or dihedral. This is the default setting for the *special_bonds* command, and means those pairwise interactions do not appear in the neighbor list. Because this fix uses a neighbor list, it also means those pairs will not be included in the RDF. This does not apply when using long-range coulomb interactions (*coul/long*, *coul/msm*, *coul/wolf* or similar. One way to get around this would be to set special_bond scaling factors to very tiny numbers that are not exactly zero (e.g., 1.0×10^{-50}). Another workaround is to write a dump file, and use the *rerun* command to compute the RDF for snapshots in the dump file. The rerun script can use a *special_bonds* command that includes all pairs in the neighbor list.

By default the RDF is computed out to the maximum force cutoff defined by the *pair_style* command. If the *cutoff* keyword is used, then the RDF is computed accurately out to the *Rcut* > 0.0 distance specified.

Note

Normally, you should only use the *cutoff* keyword if no pair style is defined (e.g., the *rerun* command is being used to post-process a dump file of snapshots) or if you really want the RDF for distances beyond the pair_style force cutoff and cannot easily post-process a dump file to calculate it. This is because using the *cutoff* keyword incurs extra computation and possibly communication, which may slow down your simulation. If you specify *Rcut* $\leq$ force cutoff, you will force an additional neighbor list to be built at every timestep this command is invoked (or every reneighboring timestep, whichever is less frequent), which is inefficient. LAMMPS will warn you if this is the case. If you specify a *Rcut* > force cutoff, you must ensure ghost atom information out to *Rcut* + *skin* is communicated, via the *comm_modify cutoff* command, else the RDF computation cannot be performed, and LAMMPS will give an error message. The *skin* value is what is specified with the *neighbor* command. In this case, you are forcing a large neighbor list to be built just for the RDF computation, and extra communication to be performed every timestep.

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

The *itypeN* and *jtypeN* settings can be specified in one of three ways. One or both of the types in the I,J pair can be a *type label*. Or an explicit numeric value can be used, as in the fourth example above. Or a wild-card asterisk can be used to specify a range 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).

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

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

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

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

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

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

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

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

3.93.4 Output info

This compute calculates a global array in which the number of rows is *Nbins* and the number of columns is $1 + 2N_{\text{pairs}}$, where N_{pairs} is the number of I, J pairings specified. The first column has the bin coordinate (center of the bin), and each successive set of two columns has the $g(r)$ and coord(r) values for a specific set of *itypeN* versus *jtypeN* interactions, as described above. These values can be used by any command that uses a global values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

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

The first column of array values will be in distance *units*. The $g(r)$ columns of array values are normalized numbers ≥ 0.0. The coordination number columns of array values are also numbers ≥ 0.0.

3.93.5 Restrictions

By default, the RDF is not computed for distances longer than the largest force cutoff, since the neighbor list creation will only contain pairs up to that distance (plus neighbor list skin). This distance can be increased using the *cutoff* keyword but this keyword is only valid with *neighbor styles 'bin' and 'nsq'*.

If you want an RDF for larger distances, you can also use the *rerun* command to post-process a dump file, use *pair style zero* and set the force cutoff to be longer in the rerun script. Note that in the rerun context, the force cutoff is arbitrary and with pair style zero you are not computing any forces, and you are not running dynamics you are not changing the model that generated the trajectory.

The definition of $g(r)$ used by LAMMPS is only appropriate for characterizing atoms that are uniformly distributed throughout the simulation cell. In such cases, the coordination number is still correct and meaningful. As an example, if a large simulation cell contains only one atom of type *itypeN* and one of *jtypeN*, then $g(r)$ will register an arbitrarily large spike at whatever distance they happen to be at, and zero everywhere else. The function coord(r) will show a step change from zero to one at the location of the spike in $g(r)$.

Note

compute rdf can handle dynamic groups and systems where atoms are added or removed, but this causes that certain normalization parameters need to be re-computed in every step and include collective communication operations. This will reduce performance and limit parallel efficiency and scaling. For systems, where only the type of atoms changes (e.g., when using *fix atom/swap*), you need to explicitly request the dynamic normalization updates via *compute_modify dynamic/dof yes*

3.93.6 Related commands

fix ave/time, *compute_modify*, *compute adf*

3.93.7 Default

The keyword defaults are cutoff = 0.0 (use the pairwise force cutoff).

3.94 compute reaxff/atom command

Accelerator Variants: *reaxff/atom/kk*

3.94.1 Syntax

```
compute ID group-ID reaxff/atom attribute args ... keyword value ...
```

- ID, group-ID are documented in *compute* command
- reaxff/atom = name of this compute command
- attribute = *pair*

  pair args = nsub
    nsub = n-instance of a sub-style, if a pair style is used multiple times in a hybrid style

- keyword = *bonds*

bonds value = no or yes
  no = ignore list of local bonds
  yes = include list of local bonds

3.94.2 Examples

```
compute 1 all reaxff/atom bonds yes
```

3.94.3 Description

Added in version 7Feb2024.

Define a computation that extracts bond information computed by the ReaxFF potential specified by *pair_style reaxff*.

By default, it produces per-atom data that includes the following columns:

- abo = atom bond order (sum of all bonds)
- nlp = number of lone pairs
- nb = number of bonds

Bonds will only be included if its atoms are in the group.

In addition, if bonds is set to yes, the compute will also produce a local array of all bonds on the current processor whose atoms are in the group. The columns of each entry of this local array are:

- id_i = atom i id of bond
- id_j = atom j id of bond
- bo = bond order of bond

3.94.4 Output info

This compute calculates a per-atom array and local array depending on the number of keywords. The number of rows in the local array is the number of bonds as described above. Both per-atom and local array have 3 columns.

The arrays can be accessed by any command that uses local and per-atom values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

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.

3.94.5 Restrictions

The compute reaxff/atom 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.

3.94.6 Related commands

pair_style reaxff

3.94.7 Default

The option defaults are *bonds* = *no*.

3.95 compute reduce command

3.96 compute reduce/region command

3.96.1 Syntax

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

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

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

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

  replace args = vec1 vec2
    vec1 = reduced value from this input vector will be replaced
    vec2 = replace it with vec1[N] where N is index of max/min value from vec2
  inputs arg = peratom or local
    peratom = all inputs are per-atom quantities (default)
    local = all input are local quantities

3.96.2 Examples

```
compute 1 all reduce sum c_force
compute 1 all reduce/region subbox sum c_force
compute 2 all reduce min c_press[2] f_ave v_myKE
compute 2 all reduce min c_press[*] f_ave v_myKE inputs peratom
compute 3 fluid reduce max c_index[1] c_index[2] c_dist replace 1 3 replace 2 3
compute 4 all reduce max c_bond inputs local
```

3.96.3 Description

Define a calculation that "reduces" one or more vector inputs into scalar values, one per listed input. For the compute reduce command, the inputs can be either per-atom or local quantities and must all be of the same kind (per-atom or local); see discussion of the optional *inputs* keyword below. The compute reduce/region command can only be used with per-atom inputs.

Atom attributes are per-atom quantities, *computes* and *fixes* can generate either per-atom or local quantities, and *atom-style variables* generate per-atom quantities. See the *variable* command and its special functions which can perform the same reduction operations as the compute reduce command on global vectors.

The reduction operation is specified by the *mode* setting. The *sum* option adds the values in the vector into a global total. The *min* or *max* options find the minimum or maximum value across all vector values. The *minabs* or *maxabs* options find the minimum or maximum value across all absolute vector values. The *ave* setting adds the vector values into a global total, then divides by the number of values in the vector. The *sumsq* option sums the square of the values in the vector into a global total. The *avesq* setting does the same as *sumsq*, then divides the sum of squares by the number of values. The last two options can be useful for calculating the variance of some quantity (e.g., variance = sumsq $-$ ave^2). The *sumabs* option sums the absolute values in the vector into a global total. The *aveabs* setting does the same as *sumabs*, then divides the sum of absolute values by the number of values.

Each listed input is operated on independently. For per-atom inputs, the group specified with this command means only atoms within the group contribute to the result. Likewise for per-atom inputs, if the compute reduce/region command is used, the atoms must also currently be within the region. Note that an input that produces per-atom quantities may define its own group which affects the quantities it returns. For example, if a compute is used as an input which generates a per-atom vector, it will generate values of 0.0 for atoms that are not in the group specified for that compute.

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

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, the following two compute reduce commands are equivalent, since the *compute stress/atom* command creates a per-atom array with six columns:

```
compute myPress all stress/atom NULL
compute 2 all reduce min c_myPress[*]
compute 2 all reduce min c_myPress[1] c_myPress[2] c_myPress[3] &
                         c_myPress[4] c_myPress[5] c_myPress[6]
```

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.

If a value begins with "c_", a compute ID must follow which has been previously defined in the input script. Valid computes can generate per-atom or local quantities. See the individual *compute* page for details. If no bracketed integer is appended, the vector calculated by the compute is used. If a bracketed integer is appended, the Ith column of the array calculated by the compute is used. Users can also write code for their own compute styles and *add them to 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. Valid fixes can generate per-atom or local quantities. See the individual *fix* page for details. Note that some fixes only produce their values on certain timesteps, which must be compatible with when compute reduce references the values, else an error results. If no bracketed integer is appended, the vector calculated by the fix is used. If a bracketed integer is appended, the Ith column of the array calculated by the fix is used. Users can also write code for their own fix style and *add them to LAMMPS*. 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. It must be an *atom-style variable*. Atom-style variables can reference thermodynamic keywords and various per-atom attributes, or invoke other computes, fixes, or variables when they are evaluated, so this is a very general means of generating per-atom quantities to reduce.

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

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

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

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

Added in version 21Nov2023.

The *inputs* keyword allows selection of whether all the inputs are per-atom or local quantities. As noted above, all the inputs must be the same kind (per-atom or local). Per-atom is the default setting. If a compute or fix is specified as an input, it must produce per-atom or local data to match this setting. If it produces both, e.g. for the *compute voronoi/atom* command, then this keyword selects between them.

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

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

3.96.4 Output info

This compute calculates a global scalar if a single input value is specified or a global vector of length *N*, where *N* is the number of inputs, and which can be accessed by indices 1 to *N*. These values can be used by any command that uses global scalar or vector values from a compute as input. See the *Howto output* doc page for an overview of LAMMPS output options.

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

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

3.96.5 Restrictions

As noted above, the compute reduce/region command can only be used with per-atom inputs.

3.96.6 Related commands

compute, *fix*, *variable*

3.96.7 Default

The default value for the *inputs* keyword is peratom.

3.97 compute reduce/chunk command

3.97.1 Syntax

```
compute ID group-ID reduce/chunk chunkID mode input1 input2 ...
```

- ID, group-ID are documented in *compute* command
- reduce/chunk = style name of this compute command
- chunkID = ID of *compute chunk/atom* command
- mode = *sum* or *min* or *max*
- one or more inputs can be listed
- input = c_ID, c_ID[N], f_ID, f_ID[N], v_ID

```
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
```

3.97.2 Examples

```
compute 1 all reduce/chunk mychunk min c_cluster
```

3.97.3 Description

Define a calculation that reduces one or more per-atom vectors into per-chunk values. This can be useful for diagnostic output. Or when used in conjunction with the *compute chunk/spread/atom* command it can be used to create per-atom values that induce a new set of chunks with a second *compute chunk/atom* command. An example is given below.

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* and *Howto chunk* doc pages for details of how chunks can be defined and examples of how they can be used to measure properties of a system.

For each atom, this compute accesses its chunk ID from the specified *chunkID* compute. The per-atom value from an input contributes to a per-chunk value corresponding the the chunk ID.

The reduction operation is specified by the *mode* setting and is performed over all the per-atom values from the atoms in each chunk. The *sum* option adds the pre-atom values to a per-chunk total. The *min* or *max* options find the minimum or maximum value of the per-atom values for each chunk.

Note that only atoms in the specified group contribute to the reduction operation. If the *chunkID* compute returns a 0 for the chunk ID of an atom (i.e., the atom is not in a chunk defined by the *compute chunk/atom* command), that atom will also not contribute to the reduction operation. An input that is a compute or fix may define its own group which affects the quantities it returns. For example, a compute with return a zero value for atoms that are not in the group specified for that compute.

Each listed input is operated on independently. Each input can be the result of a *compute* or *fix* or the evaluation of an atom-style *variable*.

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 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. For example, the following two compute reduce/chunk commands are equivalent, since the *compute property/chunk* command creates a per-atom array with 3 columns:

```
compute prop all property/atom vx vy vz
compute 10 all reduce/chunk mychunk max c_prop[*]
compute 10 all reduce/chunk mychunk max c_prop[1] c_prop[2] c_prop[3]
```

Here is an example of using this compute, in conjunction with the compute chunk/spread/atom command to identify self-assembled micelles. The commands below can be added to the examples/in.micelle script.

Imagine a collection of polymer chains or small molecules with hydrophobic end groups. All the hydrophobic (HP) atoms are assigned to a group called "phobic".

These commands will assign a unique cluster ID to all HP atoms within a specified distance of each other. A cluster will contain all HP atoms in a single molecule, but also the HP atoms in nearby molecules (e.g., molecules that have clumped to form a micelle due to the attraction induced by the hydrophobicity). The output of the chunk/reduce command will be a cluster ID per chunk (molecule). Molecules with the same cluster ID are in the same micelle.

```
group phobic type 4     # specific to in.micelle model
compute cluster phobic cluster/atom 2.0
compute cmol all chunk/atom molecule
compute reduce phobic reduce/chunk cmol min c_cluster
```

This per-chunk info could be output in at least two ways:

```
fix 10 all ave/time 1000 1 1000 c_reduce file tmp.phobic mode vector

compute spread all chunk/spread/atom cmol c_reduce
dump 1 all custom 1000 tmp.dump id type mol x y z c_cluster c_spread
dump_modify 1 sort id
```

In the first case, each snapshot in the tmp.phobic file will contain one line per molecule. Molecules with the same value are in the same micelle. In the second case each dump snapshot contains all atoms, each with a final field with the cluster ID of the micelle that the HP atoms of that atom's molecule belong to.

The result from compute chunk/spread/atom can be used to define a new set of chunks, where all the atoms in all the molecules in the same micelle are assigned to the same chunk (i.e., one chunk per micelle).

```
compute micelle all chunk/atom c_spread compress yes
```

Further analysis on a per-micelle basis can now be performed using any of the per-chunk computes listed on the *Howto chunk* doc page (e.g., count the number of atoms in each micelle, calculate its center or mass, shape/moments of inertia, and radius of gyration).

```
compute prop all property/chunk micelle count
fix 20 all ave/time 1000 1 1000 c_prop file tmp.micelle mode vector
```

Each snapshot in the tmp.micelle file will have one line per micelle with its count of atoms, plus a first line for a chunk with all the solvent atoms. By the time 50000 steps have elapsed, there are a handful of large micelles.

3.97.4 Output info

This compute calculates a global vector if a single input value is specified, otherwise a global array is output. The number of columns in the array is the number of inputs provided. The length of the vector or the number of vector elements or array rows = the number of chunks *Nchunk* as calculated by the specified *compute chunk/atom* command. The vector or array can be accessed by any command that uses global values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

The per-atom values for the vector or each column of the array will be in whatever *units* the corresponding input value is in. The vector or array values are "intensive".

3.97.5 Restrictions

none

3.97.6 Related commands

compute chunk/atom, *compute reduce*, *compute chunk/spread/atom*

3.97.7 Default

none

3.98 compute rheo/property/atom command

3.98.1 Syntax

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

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

```
possible attributes = phase, surface, surface/r,
                      surface/divr, surface/n/a, coordination,
                      shift/v/a, energy, temperature, heatflow,
                      conductivity, cv, viscosity, pressure, rho,
                      grad/v/ab, stress/v/ab, stress/t/ab, nbond/shell
```

phase = atom phase state
surface = atom surface status
surface/r = atom distance from the surface
surface/divr = divergence of position at atom position
surface/n/a = a-component of surface normal vector
coordination = coordination number
shift/v/a = a-component of atom shifting velocity
energy = atom energy
temperature = atom temperature
heatflow = atom heat flow
conductivity = atom conductivity
cv = atom specific heat
viscosity = atom viscosity
pressure = atom pressure
rho = atom density
grad/v/ab = ab-component of atom velocity gradient tensor
stress/v/ab = ab-component of atom viscous stress tensor
stress/t/ab = ab-component of atom total stress tensor (pressure and viscous)
nbond/shell = number of oxide bonds

3.98.2 Examples

```
compute 1 all rheo/property/atom phase surface/r surface/n/* pressure
compute 2 all rheo/property/atom shift/v/x grad/v/xx stress/v/*
```

3.98.3 Description

Added in version 29Aug2024.

Define a computation that stores atom attributes specific to the RHEO package for each atom in the group. This is useful so that the values can be used by other *output commands* that take computes as inputs. See for example, the *compute reduce*, *fix ave/atom*, *fix ave/histo*, *fix ave/chunk*, and *atom-style variable* commands.

For vector attributes, e.g. *shift/v/*α, one must specify α as the *x*, *y*, or *z* component, e.g. *shift/v/x*. Alternatively, a wild card * will include all components, *x* and *y* in 2D or *x*, *y*, and *z* in 3D.

For tensor attributes, e.g. *grad/v/*$\alpha\beta$, one must specify both α and β as *x*, *y*, or *z*, e.g. *grad/v/xy*. Alternatively, a wild card * will include all components. In 2D, this includes *xx*, *xy*, *yx*, and *yy*. In 3D, this includes *xx*, *xy*, *xz*, *yx*, *yy*, *yz*, *zx*, *zy*, and *zz*.

Many properties require their respective fixes, listed below in related commands, be defined. For instance, the *viscosity* attribute is the viscosity of a particle calculated by *fix rheo/viscous*. The meaning of less obvious properties is described below.

The *phase* property indicates whether the particle is in a fluid state, a value of 0, or a solid state, a value of 1.

The *surface* property indicates the surface designation produced by the *interface/reconstruct* option of *fix rheo*. Bulk particles have a value of 0, surface particles have a value of 1, and splash particles have a value of 2. The *surface/r* property is the distance from the surface, up to the kernel cutoff length. Surface particles have a value of 0. The *surface/n/*α properties are the components of the surface normal vector.

The *shift/v/*α properties are the components of the shifting velocity produced by the *shift* option of *fix rheo*.

The *nbond/shell* property is the number of shell bonds that have been activated from *bond style rheo/shell*.

The values are stored in a per-atom vector or array as discussed below. Zeroes are stored for atoms not in the specified group or for quantities that are not defined for a particular particle in the group

3.98.4 Output info

This compute calculates a per-atom vector or per-atom array depending on the number of input values. Generally, if a single input is specified, a per-atom vector is produced. If two or more inputs are specified, a per-atom array is produced where the number of columns = the number of inputs. However, if a wild card * is used for a vector or tensor, then the number of inputs is considered to be incremented by the dimension or the dimension squared, respectively. The vector or array can be accessed by any command that uses per-atom values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

The vector or array values will be in whatever *units* the corresponding attribute is in (e.g., density units for *rho*).

3.98.5 Restrictions

none

3.98.6 Related commands

dump custom, *compute reduce*, *fix ave/atom*, *fix ave/chunk*, *fix rheo/viscosity*, *fix rheo/pressure*, *fix rheo/thermal*, *fix rheo/oxdiation*, *fix rheo*

3.98.7 Default

none

3.99 compute rigid/local command

3.99.1 Syntax

```
compute ID group-ID rigid/local rigidID input1 input2 ...
```

- ID, group-ID are documented in *compute* command
- rigid/local = style name of this compute command
- rigidID = ID of fix rigid/small command or one of its variants
- input = one or more rigid body attributes

```
possible attributes = id, mol, mass,
                      x, y, z, xu, yu, zu, ix, iy, iz
                      vx, vy, vz, fx, fy, fz,
                      omegax, omegay, omegaz,
                      angmomx, angmomy, angmomz,
                      quatw, quati, quatj, quatk,
                      tqx, tqy, tqz,
                      inertiax, inertiay, inertiaz
```

id = atom ID of atom within body which owns body properties
mol = molecule ID used to define body in fix rigid/small command
mass = total mass of body
x,y,z = center of mass coords of body
xu,yu,zu = unwrapped center of mass coords of body
ix,iy,iz = box image that the center of mass is in
vx,vy,vz = center of mass velocities
fx,fy,fz = force of center of mass
omegax,omegay,omegaz = angular velocity of body
angmomx,angmomy,angmomz = angular momentum of body
quatw,quati,quatj,quatk = quaternion components for body
tqx,tqy,tqz = torque on body
inertiax,inertiay,inertiaz = diagonalized moments of inertia of body

3.99.2 Examples

```
compute 1 all rigid/local myRigid mol x y z
```

3.99.3 Description

Define a computation that simply stores rigid body attributes for rigid bodies defined by the *fix rigid/small* command or one of its NVE, NVT, NPT, NPH variants. The data is stored as local data so it can be accessed by other *output commands* that process local data, such as the *compute reduce* or *dump local* commands.

Note that this command only works with the *fix rigid/small* command or its variants, not the fix rigid command and its variants. The ID of the *fix rigid/small* command used to define rigid bodies must be specified as *rigidID*. The *fix rigid* command is typically used to define a handful of (potentially very large) rigid bodies. It outputs similar per-body information as this command directly from the fix as global data; see the *fix rigid* page for details

The local data stored by this command is generated by looping over all the atoms owned on a processor. If the atom is not in the specified *group-ID* or is not part of a rigid body it is skipped. If it is not the atom within a body that is assigned to store the body information it is skipped (only one atom per body is so assigned). If it is the assigned atom, then the info for that body is output. This means that information for *N* bodies is generated. *N* may be less than the number of bodies defined by the fix rigid command, if the atoms in some bodies are not in the *group-ID*.

Note

Which atom in a body owns the body info is determined internal to LAMMPS; it's the one nearest the geometric center of the body. Typically you should avoid this complication, by defining the group associated with this fix to include/exclude entire bodies.

Note that as atoms and bodies migrate from processor to processor, there will be no consistent ordering of the entries within the local vector or array from one timestep to the next.

Here is an example of how to use this compute to dump rigid body info to a file:

```
compute 1 all rigid/local myRigid mol x y z fx fy fz
dump 1 all local 1000 tmp.dump index c_1[1] c_1[2] c_1[3] c_1[4] c_1[5] c_1[6] c_1[7]
```

This section explains the rigid body attributes that can be specified.

The *id* attribute is the atom-ID of the atom which owns the rigid body, which is assigned by the *fix rigid/small* command.

The *mol* attribute is the molecule ID of the rigid body. It should be the molecule ID which all of the atoms in the body belong to, since that is how the *fix rigid/small* command defines its rigid bodies.

The *mass* attribute is the total mass of the rigid body.

There are two options for outputting the coordinates of the center of mass (COM) of the body. The *x*, *y*, *z* attributes write the COM "unscaled", in the appropriate distance *units* (Å, σ, etc). Use *xu*, *yu*, *zu* if you want the COM "unwrapped" by the image flags for each body. Unwrapped means that if the body COM has passed through a periodic boundary one or more times, the value is generated what the COM coordinate would be if it had not been wrapped back into the periodic box.

The image flags for the body can be generated directly using the *ix*, *iy*, *iz* attributes. For periodic dimensions, they specify which image of the simulation box the COM 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 body COMs cross periodic boundaries during the simulation.

The *vx*, *vy*, *vz*, *fx*, *fy*, *fz* attributes are components of the COM velocity and force on the COM of the body.

The *omegax*, *omegay*, and *omegaz* attributes are the angular velocity components of the body in the system frame around its COM.

The *angmomx*, *angmomy*, and *angmomz* attributes are the angular momentum components of the body in the system frame around its COM.

The *quatw*, *quati*, *quatj*, and *quatk* attributes are the components of the 4-vector quaternion representing the orientation of the rigid body. See the *set* command for an explanation of the quaternion vector.

The *tqx*, *tqy*, *tqz* attributes are components of the torque acting on the body around its COM.

The *inertiax*, *inertiay*, *inertiaz* attributes are components of diagonalized inertia tensor for the body (i.e., the three moments of inertia for the body around its principal axes), as computed internally by LAMMPS.

3.99.4 Output info

This compute calculates a local vector or local array depending on the number of keywords. The length of the vector or number of rows in the array is the number of rigid bodies. If a single keyword is specified, a local vector is produced. If two or more keywords are specified, a local array is produced where the number of columns = the number of keywords. The vector or array can be accessed by any command that uses local values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

The vector or array values will be in whatever *units* the corresponding attribute is in:

- id,mol = unitless
- mass = mass units
- x,y,z and xy,yu,zu = distance units
- vx,vy,vz = velocity units
- fx,fy,fz = force units
- omegax,omegay,omegaz = radians/time units
- angmomx,angmomy,angmomz = mass*distance^2/time units
- quatw,quati,quatj,quatk = unitless
- tqx,tqy,tqz = torque units
- inertiax,inertiay,inertiaz = mass*distance^2 units

3.99.5 Restrictions

This compute 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.

3.99.6 Related commands

dump local, *compute reduce*

3.99.7 Default

none

3.100 compute saed command

3.100.1 Syntax

```
compute ID group-ID saed lambda type1 type2 ... typeN keyword value ...
```

- ID, group-ID are documented in *compute* command
- saed = style name of this compute command
- lambda = wavelength of incident radiation (length units)
- type1 type2 ... typeN = chemical symbol of each atom type (see valid options below)
- zero or more keyword/value pairs may be appended
- keyword = *Kmax* or *Zone* or *dR_Ewald* or *c* or *manual* or *echo*

  Kmax value = Maximum distance explored from reciprocal space origin
  (inverse length units)
  Zone values = z1 z2 z3
  z1,z2,z3 = Zone axis of incident radiation. If z1=z2=z3=0 all
  reciprocal space will be meshed up to Kmax
  dR_Ewald value = Thickness of Ewald sphere slice intercepting
  reciprocal space (inverse length units)
  c values = c1 c2 c3
  c1,c2,c3 = parameters to adjust the spacing of the reciprocal
  lattice nodes in the h, k, and l directions respectively
  manual = flag to use manual spacing of reciprocal lattice points
  based on the values of the c parameters
  echo = flag to provide extra output for debugging purposes

3.100.2 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
```

3.100.3 Description

Define a computation that calculates electron diffraction intensity as described in *(Coleman)* on a mesh of reciprocal lattice nodes defined by the entire simulation domain (or manually) using simulated radiation of wavelength lambda.

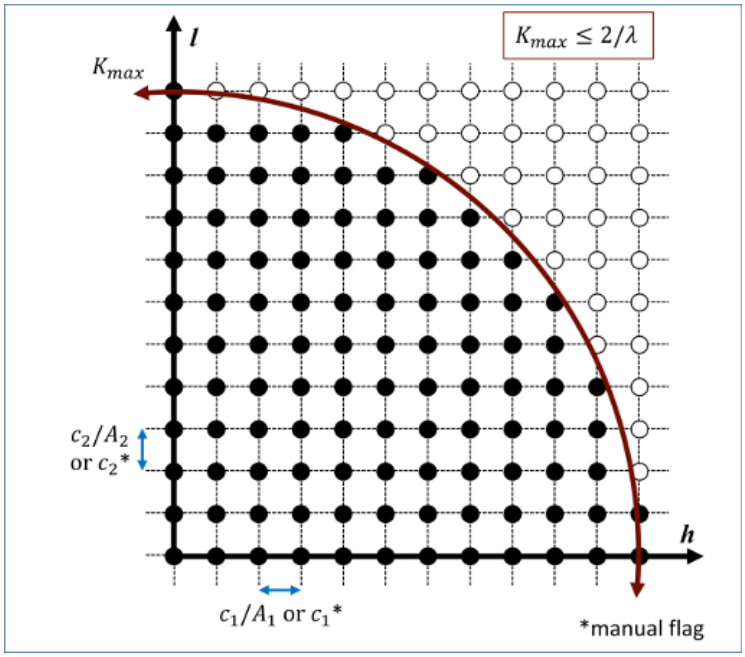
The electron diffraction intensity I at each reciprocal lattice point is computed from the structure factor F using the equations:

$$I = \frac{F^* F}{N}$$

$$F(\mathbf{k}) = \sum_{j=1}^{N} f_j(\theta) exp(2\pi i \mathbf{k} \cdot \mathbf{r}_j)$$

Here, K is the location of the reciprocal lattice node, r_j is the position of each atom, f_j are atomic scattering factors.

Diffraction intensities are calculated on a three-dimensional mesh of reciprocal lattice nodes. The mesh spacing is defined either (a) by the entire simulation domain or (b) manually using selected values as shown in the 2D diagram below.

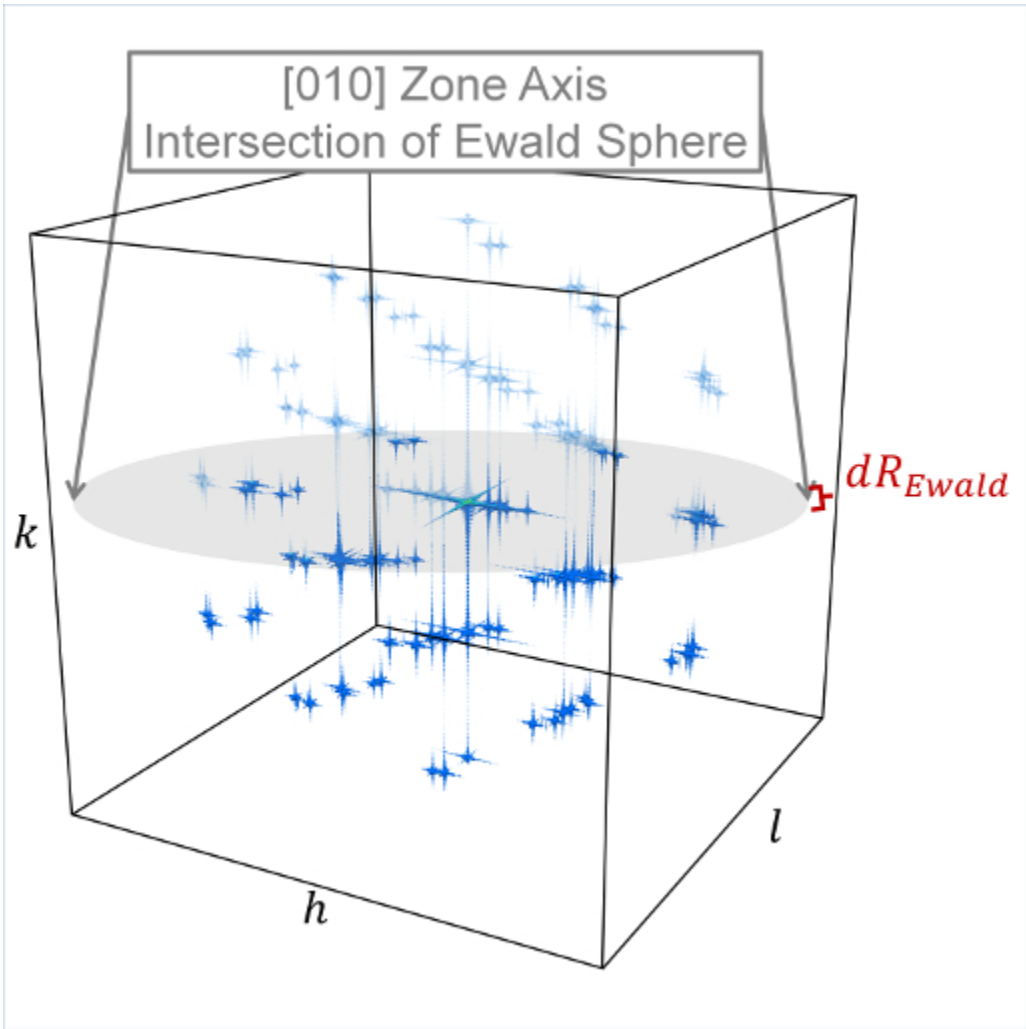


For a mesh defined by the simulation domain, a rectilinear grid is constructed with spacing *c*inv(A) along each reciprocal lattice axis. Where A are the vectors corresponding to the edges of the simulation cell. If one or two directions has non-periodic boundary conditions, then the spacing in these directions is defined from the average of the (inversed) box lengths with periodic boundary conditions. Meshes defined by the simulation domain must contain at least one periodic boundary.

If the *manual* flag is included, the mesh of reciprocal lattice nodes will defined using the *c* values for the spacing along each reciprocal lattice axis. Note that manual mapping of the reciprocal space mesh is good for comparing diffraction results from multiple simulations; however it can reduce the likelihood that Bragg reflections will be satisfied unless small spacing parameters (< 0.05 Å$^{-}1$) are implemented. Meshes with manual spacing do not require a periodic boundary.

The limits of the reciprocal lattice mesh are determined by the use of the *Kmax*, *Zone*, and *dR_Ewald* parameters. The rectilinear mesh created about the origin of reciprocal space is terminated at the boundary of a sphere of radius *Kmax* centered at the origin. If *Zone* parameters *z1* = *z2* = *z3* = 0 are used, diffraction intensities are computed throughout the entire spherical volume - note this can greatly increase the cost of computation. Otherwise, *Zone* parameters will denote the $z1 = h$, $z2 = k$, and $z3 = \ell$ (in a global sense) zone axis of an intersecting Ewald sphere. Diffraction

intensities will only be computed at the intersection of the reciprocal lattice mesh and a *dR_Ewald* thick surface of the Ewald sphere. See the example 3D intensity data and the intersection of a [010] zone axis in the below image.



The atomic scattering factors, fj, accounts for the reduction in diffraction intensity due to Compton scattering. Compute saed uses analytical approximations of the atomic scattering factors that vary for each atom type (type1 type2 ... typeN) and angle of diffraction. The analytic approximation is computed using the formula *(Brown)*:

$$f_j\left(\frac{sin(\theta)}{\lambda}\right) = \sum_i^5 a_i exp\left(-b_i \frac{sin^2(\theta)}{\lambda^2}\right)$$

Coefficients parameterized by *(Fox)* are assigned for each atom type designating the chemical symbol and charge of each atom type. Valid chemical symbols for compute saed are:

H	He	Li	Be	B	C	N	O	F	Ne	Na	Mg	Al	Si	P	S	Cl	Ar	K	Ca
Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Cu	Zn	Ga	Ge	As	Se	Br	Kr	Rb	Sr	Y	Zr
Nb	Mo	Tc	Ru	Rh	Pd	Ag	Cd	In	Sn	Sb	Te	I	Xe	Cs	Ba	La	Ce	Pr	Nd
Pm	Sm	Eu	Gd	Tb	Dy	Ho	Er	Tm	Yb	Lu	Hf	Ta	W	Re	Os	Ir	Pt	Au	Hg
Tl	Pb	Bi	Po	At	Rn	Fr	Ra	Ac	Th	Pa	U	Np	Pu	Am	Cm	Bk	Cf		

If the *echo* keyword is specified, compute saed will provide extra reporting information to the screen.

3.100.4 Output info

This compute calculates a global vector. The length of the vector is the number of reciprocal lattice nodes that are explored by the mesh. The entries of the global vector are the computed diffraction intensities as described above.

The vector can be accessed by any command that uses global values from a compute as input. See the *Howto output* doc page for an overview of LAMMPS output options.

All array values calculated by this compute are "intensive".

3.100.5 Restrictions

This compute 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.

The compute_saed command does not work for triclinic cells.

3.100.6 Related commands

fix saed_vtk, *compute xrd*

3.100.7 Default

The option defaults are Kmax = 1.70, Zone 1 0 0, c 1 1 1, dR_Ewald = 0.01.

(Coleman) Coleman, Spearot, Capolungo, MSMSE, 21, 055020 (2013).

(Brown) Brown et al. International Tables for Crystallography Volume C: Mathematical and Chemical Tables, 554-95 (2004).

(Fox) Fox, O'Keefe, Tabbernor, Acta Crystallogr. A, 45, 786-93 (1989).

3.101 compute slcsa/atom command

3.101.1 Syntax

```
compute ID group-ID slcsa/atom twojmax nclasses db_mean_descriptor_file lda_file lr_decision_file lr_
↪bias_file maha_file value
```

- ID, group-ID are documented in *compute* command
- slcsa/atom = style name of this compute command
- twojmax = band limit for bispectrum components (non-negative integer)
- nclasses = number of crystal structures used in the database for the classifier SL-CSA
- db_mean_descriptor_file = file name of file containing the database mean descriptor
- lda_file = file name of file containing the linear discriminant analysis matrix for dimension reduction
- lr_decision_file = file name of file containing the scaling matrix for logistic regression classification
- lr_bias_file = file name of file containing the bias vector for logistic regression classification

- maha_file = file name of file containing for each crystal structure: the Mahalanobis distance threshold for sanity check purposes, the average reduced descriptor and the inverse of the corresponding covariance matrix
- c_ID[*] = compute ID of previously required *compute sna/atom* command

3.101.2 Examples

```
compute b1 all sna/atom 9.0 0.99363 8 0.5 1.0 rmin0 0.0 nnn 24 wmode 1 delta 0.3
compute b2 all slcsa/atom 8 4 mean_descriptors.dat lda_scalings.dat lr_decision.dat lr_bias.dat maha_
→thresholds.dat c_b1[*]
```

3.101.3 Description

Added in version 7Feb2024.

Define a computation that performs the Supervised Learning Crystal Structure Analysis (SL-CSA) from *(Lafourcade)* for each atom in the group. The SL-CSA tool takes as an input a per-atom descriptor (bispectrum) that is computed through the *compute sna/atom* command and then proceeds to a dimension reduction step followed by a logistic regression in order to assign a probable crystal structure to each atom in the group. The SL-CSA tool is pre-trained on a database containing C distinct crystal structures from which a crystal structure classifier is derived and a tutorial to build such a tool is available at SL-CSA.

The first step of the SL-CSA tool consists in performing a dimension reduction of the per-atom descriptor $\mathbf{B}^i \in \mathbb{R}^D$ through the Linear Discriminant Analysis (LDA) method, leading to a new projected descriptor $\mathbf{x}^i = \mathrm{P}_{\mathrm{LDA}}(\mathbf{B}^i) : \mathbb{R}^D \rightarrow \mathbb{R}^{d=C-1}$:

$$\mathbf{x}^i = \mathbf{C}^T_{\mathrm{LDA}} \cdot (\mathbf{B}^i - \mu^{\mathbf{B}}_{\mathrm{db}})$$

where $\mathbf{C}^T_{\mathrm{LDA}} \in \mathbb{R}^{D\times d}$ is the reduction coefficients matrix of the LDA model read in file *lda_file*, $\mathbf{B}^i \in \mathbb{R}^D$ is the bispectrum of atom i and $\mu^{\mathbf{B}}_{\mathrm{db}} \in \mathbb{R}^D$ is the average descriptor of the entire database. The latter is computed from the average descriptors of each crystal structure read from the file *mean_descriptors_file*.

The new projected descriptor with dimension $d = C - 1$ allows for a good separation of different crystal structures fingerprints in the latent space.

Once the dimension reduction step is performed by means of LDA, the new descriptor $\mathbf{x}^i \in \mathbb{R}^{d=C-1}$ is taken as an input for performing a multinomial logistic regression (LR) which provides a score vector $\mathbf{s}^i = \mathrm{P}_{\mathrm{LR}}(\mathbf{x}^i) : \mathbb{R}^d \rightarrow \mathbb{R}^C$ defined as:

$$\mathbf{s}^i = \mathbf{b}_{\mathrm{LR}} + \mathbf{D}_{\mathrm{LR}} \cdot \mathbf{x}^{iT}$$

with $\mathbf{b}_{\mathrm{LR}} \in \mathbb{R}^C$ and $\mathbf{D}_{\mathrm{LR}} \in \mathbb{R}^{C\times d}$ the bias vector and decision matrix of the LR model after training both read in files *lr_fil1* and *lr_file2* respectively.

Finally, a probability vector $\mathbf{p}^i = \mathrm{P}_{\mathrm{LR}}(\mathbf{x}^i) : \mathbb{R}^d \rightarrow \mathbb{R}^C$ is defined as:

$$\mathbf{p}^i = \frac{\exp(\mathbf{s}^i)}{\sum_j \exp(s^i_j)}$$

from which the crystal structure assigned to each atom with descriptor $\mathbf{B}^i$ and projected descriptor $\mathbf{x}^i$ is computed as the *argmax* of the probability vector $\mathbf{p}^i$. Since the logistic regression step systematically attributes a crystal structure to each atom, a sanity check is needed to avoid misclassification. To this end, a per-atom Mahalanobis distance to each crystal structure CS present in the database is computed:

$$d^{i\rightarrow \mathrm{CS}}_{\mathrm{Mahalanobis}} = \sqrt{(\mathbf{x}^i - \mu^{\mathbf{x}}_{\mathrm{CS}})^{\mathrm{T}} \cdot \blacksquare^{-1}_{\mathrm{CS}} \cdot (\mathbf{x}^i - \mu^{\mathbf{x}}_{\mathrm{CS}})}$$

where $\mu^{\mathbf{x}}_{\mathrm{CS}} \in \mathbb{R}^d$ is the average projected descriptor of crystal structure *CS* in the database and where $\blacksquare_{\mathrm{CS}} \in \mathbb{R}^{d \times d}$ is the corresponding covariance matrix. Finally, if the Mahalanobis distance to crystal structure *CS* for atom *i* is greater than the pre-determined threshold, no crystal structure is assigned to atom *i*. The Mahalanobis distance thresholds are read in file *maha_file* while the covariance matrices are read in file *covmat_file*.

The SL-CSA framework provides an automatic computation of the different matrices and thresholds required for a proper classification and writes down all the required files for calling the *compute slcsa/atom* command.

The *compute slcsa/atom* command requires that the *compute sna/atom* command is called before as it takes the resulting per-atom bispectrum as an input. In addition, it is crucial that the value *twojmax* is set to the same value of the value *twojmax* used in the *compute sna/atom* command, as well as that the value *nclasses* is set to the number of crystal structures used in the database to train the SL-CSA tool.

3.101.4 Output info

By default, this compute computes the Mahalanobis distances to the different crystal structures present in the database in addition to assigning a crystal structure for each atom as a per-atom vector, which can be accessed by any command that uses per-atom values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

3.101.5 Restrictions

This compute is part of the EXTRA-COMPUTE package. It is only enabled if LAMMPS was built with that package. See the *Build package* page for more info.

3.101.6 Related commands

compute sna/atom

3.101.7 Default

none

(Lafourcade) Lafourcade, Maillet, Denoual, Duval, Allera, Goryaeva, and Marinica, Comp. Mat. Science, 230, 112534 (2023)

3.102 compute slice command

3.102.1 Syntax

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

- ID, group-ID are documented in *compute* command
- slice = style name of this compute command
- Nstart = starting index within input vector(s)
- Nstop = stopping index within input vector(s)

- Nskip = extract every Nskip elements from input vector(s)
- input = c_ID, c_ID[N], f_ID, f_ID[N]

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

3.102.2 Examples

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

3.102.3 Description

Define a calculation that "slices" one or more vector inputs into smaller vectors, one per listed input. The inputs can be global quantities; they cannot be per-atom or local quantities. *Computes* and *fixes* and vector-style *variables* can generate such global quantities. The group specified with this command is ignored.

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

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

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

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

If an input value begins with "v_", a variable name must follow which has been previously defined in the input script. Only vector-style variables can be referenced. See the *variable* command for details. Note that variables of style *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 slice.

If a single input is specified this compute produces a global vector, even if the length of the vector is 1. If multiple inputs are specified, then a global array of values is produced, with the number of columns equal to the number of inputs specified.

3.102.4 Output info

This compute calculates a global vector if a single input value is specified or a global array with N columns where N is the number of inputs. The length of the vector or the number of rows in the array is equal to the number of values extracted from each input vector. These values can be used by any command that uses global vector or array values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

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

The vector or array values will be in whatever *units* the input quantities are in.

3.102.5 Restrictions

none

3.102.6 Related commands

compute, *fix*, *compute reduce*

3.102.7 Default

none

3.103 compute smd/contact/radius command

3.103.1 Syntax

```
compute ID group-ID smd/contact/radius
```

- ID, group-ID are documented in *compute* command
- smd/contact/radius = style name of this compute command

3.103.2 Examples

```
compute 1 all smd/contact/radius
```

3.103.3 Description

Define a computation which outputs the contact radius, i.e., the radius used to prevent particles from penetrating each other. The contact radius is used only to prevent particles belonging to different physical bodies from penetrating each other. It is used by the contact pair styles, e.g., smd/hertz and smd/tri_surface.

See this PDF guide to using Smooth Mach Dynamics in LAMMPS.

The value of the contact radius will be 0.0 for particles not in the specified compute group.

3.103.4 Output info

This compute calculates a per-particle vector, which can be accessed by any command that uses per-particle values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

The per-particle vector values will be in distance *units*.

3.103.5 Restrictions

This compute 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.

3.103.6 Related commands

dump custom smd/hertz smd/tri_surface

3.103.7 Default

none

3.104 compute smd/damage command

3.104.1 Syntax

```
compute ID group-ID smd/damage
```

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

3.104.2 Examples

```
compute 1 all smd/damage
```

3.104.3 Description

Define a computation that calculates the damage status of SPH particles according to the damage model which is defined via the SMD SPH pair styles, e.g., the maximum plastic strain failure criterion.

See this PDF guide to use Smooth Mach Dynamics in LAMMPS.

Output Info:

This compute calculates a per-particle vector, which can be accessed by any command that uses per-particle values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

The per-particle values are dimensionless an in the range of zero to one.

3.104.4 Restrictions

This compute is part of the MACHDYN package. It is only enabled if LAMMPS was built with that package. See the "Build

3.104.5 Related commands

smd/plastic_strain, *smd/tlsph_stress*

3.104.6 Default

none

3.105 compute smd/hourglass/error command

3.105.1 Syntax

```
compute ID group-ID smd/hourglass/error
```

- ID, group-ID are documented in *compute* command
- smd/hourglass/error = style name of this compute command

3.105.2 Examples

```
compute 1 all smd/hourglass/error
```

3.105.3 Description

Define a computation which outputs the error of the approximated relative separation with respect to the actual relative separation of the particles i and j. Ideally, if the deformation gradient is exact, and there exists a unique mapping between all particles' positions within the neighborhood of the central node and the deformation gradient, the approximated relative separation will coincide with the actual relative separation of the particles i and j in the deformed configuration. This compute is only really useful for debugging the hourglass control mechanism which is part of the Total-Lagrangian SPH pair style.

See this PDF guide to use Smooth Mach Dynamics in LAMMPS.

Output Info:

This compute calculates a per-particle vector, which can be accessed by any command that uses per-particle values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

The per-particle vector values will are dimensionless. See *units*.

3.105.4 Restrictions

This compute 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.

This quantity will be computed only for particles which interact with tlsph pair style.

3.105.5 Related commands

smd/tlsph_defgrad

3.105.6 Default

3.106 compute smd/internal/energy command

3.106.1 Syntax

```
compute ID group-ID smd/internal/energy
```

- ID, group-ID are documented in *compute* command
- smd/smd/internal/energy = style name of this compute command

3.106.2 Examples

```
compute 1 all smd/internal/energy
```

3.106.3 Description

Define a computation which outputs the per-particle enthalpy, i.e., the sum of potential energy and heat.

See this PDF guide to use Smooth Mach Dynamics in LAMMPS.

3.106.4 Output Info

This compute calculates a per-particle vector, which can be accessed by any command that uses per-particle values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

The per-particle vector values will be given in *units* of energy.

3.106.5 Restrictions

This compute 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. This compute can only be used for particles which interact via the updated Lagrangian or total Lagrangian SPH pair styles.

3.106.6 Related commands

none

3.106.7 Default

3.107 compute smd/plastic/strain command

3.107.1 Syntax

```
compute ID group-ID smd/plastic/strain
```

- ID, group-ID are documented in *compute* command
- smd/plastic/strain = style name of this compute command

3.107.2 Examples

```
compute 1 all smd/plastic/strain
```

3.107.3 Description

Define a computation that outputs the equivalent plastic strain per particle. This command is only meaningful if a material model with plasticity is defined.

See this PDF guide to use Smooth Mach Dynamics in LAMMPS.

Output Info:

This compute calculates a per-particle vector, which can be accessed by any command that uses per-particle values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

The per-particle values will be given dimensionless. See *units*.

3.107.4 Restrictions

This compute 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. This compute can only be used for particles which interact via the updated Lagrangian or total Lagrangian SPH pair styles.

3.107.5 Related commands

smd/plastic/strain/rate, *smd/tlsph/strain/rate*, *smd/tlsph/strain*

3.107.6 Default

none

3.108 compute smd/plastic/strain/rate command

3.108.1 Syntax

```
compute ID group-ID smd/plastic/strain/rate
```

- ID, group-ID are documented in *compute* command
- smd/plastic/strain/rate = style name of this compute command

3.108.2 Examples

```
compute 1 all smd/plastic/strain/rate
```

3.108.3 Description

Define a computation that outputs the time rate of the equivalent plastic strain. This command is only meaningful if a material model with plasticity is defined.

See this PDF guide to use Smooth Mach Dynamics in LAMMPS.

Output Info:

This compute calculates a per-particle vector, which can be accessed by any command that uses per-particle values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

The per-particle values will be given in *units* of one over time.

3.108.4 Restrictions

This compute 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. This compute can only be used for particles which interact via the updated Lagrangian or total Lagrangian SPH pair styles.

3.108.5 Related commands

smd/plastic/strain, *smd/tlsph/strain/rate*, *smd/tlsph/strain*

3.108.6 Default

none

3.109 compute smd/rho command

3.109.1 Syntax

```
compute ID group-ID smd/rho
```

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

3.109.2 Examples

```
compute 1 all smd/rho
```

3.109.3 Description

Define a computation that calculates the per-particle mass density. The mass density is the mass of a particle which is constant during the course of a simulation, divided by its volume, which can change due to mechanical deformation.

See this PDF guide to use Smooth Mach Dynamics in LAMMPS.

3.109.4 Output info

This compute calculates a per-particle vector, which can be accessed by any command that uses per-particle values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

The per-particle values will be in *units* of mass over volume.

3.109.5 Restrictions

This compute 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.

3.109.6 Related commands

compute smd/vol

3.109.7 Default

none

3.110 compute smd/tlsph/defgrad command

3.110.1 Syntax

```
compute ID group-ID smd/tlsph/defgrad
```

- ID, group-ID are documented in *compute* command
- smd/tlsph/defgrad = style name of this compute command

3.110.2 Examples

```
compute 1 all smd/tlsph/defgrad
```

3.110.3 Description

Define a computation that calculates the deformation gradient. It is only meaningful for particles which interact according to the Total-Lagrangian SPH pair style.

See this PDF guide to use Smooth Mach Dynamics in LAMMPS.

3.110.4 Output info

This compute outputs a per-particle vector of vectors (tensors), which can be accessed by any command that uses per-particle values from a compute as input. See the *Howto output* doc page for an overview of LAMMPS output options.

The per-particle vector values will be given dimensionless. See *units*. The per-particle vector has 10 entries. The first nine entries correspond to the xx, xy, xz, yx, yy, yz, zx, zy, zz components of the asymmetric deformation gradient tensor. The tenth entry is the determinant of the deformation gradient.

3.110.5 Restrictions

This compute 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. TThis compute can only be used for particles which interact via the total Lagrangian SPH pair style.

3.110.6 Related commands

smd/hourglass/error

3.110.7 Default

none

3.111 compute smd/tlsph/dt command

3.111.1 Syntax

```
compute ID group-ID smd/tlsph/dt
```

- ID, group-ID are documented in *compute* command
- smd/tlsph/dt = style name of this compute command

3.111.2 Examples

```
compute 1 all smd/tlsph/dt
```

3.111.3 Description

Define a computation that outputs the CFL-stable time increment per particle. This time increment is essentially given by the speed of sound, divided by the SPH smoothing length. Because both the speed of sound and the smoothing length typically change during the course of a simulation, the stable time increment needs to be re-computed every time step. This calculation is performed automatically in the relevant SPH pair styles and this compute only serves to make the stable time increment accessible for output purposes.

See this PDF guide to using Smooth Mach Dynamics in LAMMPS.

3.111.4 Output info

This compute calculates a per-particle vector, which can be accessed by any command that uses per-particle values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

The per-particle values will be given in *units* of time.

3.111.5 Restrictions

This compute 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.

This compute can only be used for particles interacting with the Total-Lagrangian SPH pair style.

3.111.6 Related commands

smd/adjust/dt

3.111.7 Default

none

3.112 compute smd/tlsph/num/neighs command

3.112.1 Syntax

```
compute ID group-ID smd/tlsph/num/neighs
```

- ID, group-ID are documented in *compute* command
- smd/tlsph/num/neighs = style name of this compute command

3.112.2 Examples

```
compute 1 all smd/tlsph/num/neighs
```

3.112.3 Description

Define a computation that calculates the number of particles inside of the smoothing kernel radius for particles interacting via the Total-Lagrangian SPH pair style.

See this PDF guide to using Smooth Mach Dynamics in LAMMPS.

3.112.4 Output info

This compute calculates a per-particle vector, which can be accessed by any command that uses per-particle values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

The per-particle values are dimensionless. See *units*.

3.112.5 Restrictions

This compute 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.

This quantity will be computed only for particles which interact with the Total-Lagrangian pair style.

3.112.6 Related commands

smd/ulsph/num/neighs

3.112.7 Default

none

3.113 compute smd/tlsph/shape command

3.113.1 Syntax

```
compute ID group-ID smd/tlsph/shape
```

- ID, group-ID are documented in *compute* command
- smd/tlsph/shape = style name of this compute command

3.113.2 Examples

```
compute 1 all smd/tlsph/shape
```

3.113.3 Description

Define a computation that outputs the current shape of the volume associated with a particle as a rotated ellipsoid. It is only meaningful for particles which interact according to the Total-Lagrangian SPH pair style.

See this PDF guide to use Smooth Mach Dynamics in LAMMPS.

3.113.4 Output info

This compute calculates a per-particle vector of vectors, which can be accessed by any command that uses per-particle values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

The per-particle vector has 7 entries. The first three entries correspond to the lengths of the ellipsoid's axes and have units of length. These axis values are computed as the contact radius times the xx, yy, or zz components of the Green-Lagrange strain tensor associated with the particle. The next 4 values are quaternions (order: q, x, y, z) which describe the spatial rotation of the particle relative to its initial state.

3.113.5 Restrictions

This compute 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.

This quantity will be computed only for particles which interact with the Total-Lagrangian SPH pair style.

3.113.6 Related commands

smd/contact/radius

3.113.7 Default

none

3.114 compute smd/tlsph/strain command

3.114.1 Syntax

```
compute ID group-ID smd/tlsph/strain
```

- ID, group-ID are documented in *compute* command
- smd/tlsph/strain = style name of this compute command

3.114.2 Examples

```
compute 1 all smd/tlsph/strain
```

3.114.3 Description

Define a computation that calculates the Green-Lagrange strain tensor for particles interacting via the Total-Lagrangian SPH pair style.

See this PDF guide to using Smooth Mach Dynamics in LAMMPS.

3.114.4 Output info

This compute calculates a per-particle vector of vectors (tensors), which can be accessed by any command that uses per-particle values from a compute as input. See the *Howto output* doc page for an overview of LAMMPS output options.

The per-particle tensor values will be given dimensionless. See *units*.

The per-particle vector has 6 entries, corresponding to the xx, yy, zz, xy, xz, yz components of the symmetric strain tensor.

3.114.5 Restrictions

This compute 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.

This quantity will be computed only for particles which interact with the Total-Lagrangian SPH pair style.

3.114.6 Related commands

smd/tlsph/strain/rate, *smd/tlsph/stress*

3.114.7 Default

none

3.115 compute smd/tlsph/strain/rate command

3.115.1 Syntax

```
compute ID group-ID smd/tlsph/strain/rate
```

- ID, group-ID are documented in *compute* command
- smd/tlsph/strain/rate = style name of this compute command

3.115.2 Examples

```
compute 1 all smd/tlsph/strain/rate
```

3.115.3 Description

Define a computation that calculates the rate of the strain tensor for particles interacting via the Total-Lagrangian SPH pair style.

See this PDF guide to using Smooth Mach Dynamics in LAMMPS.

3.115.4 Output info

This compute calculates a per-particle vector of vectors (tensors), which can be accessed by any command that uses per-particle values from a compute as input. See the *Howto output* doc page for an overview of LAMMPS output options.

The values will be given in *units* of one over time.

The per-particle vector has 6 entries, corresponding to the xx, yy, zz, xy, xz, yz components of the symmetric strain rate tensor.

3.115.5 Restrictions

This compute 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.

This quantity will be computed only for particles which interact with Total-Lagrangian SPH pair style.

3.115.6 Related commands

compute smd/tlsph/strain, *compute smd/tlsph/stress*

3.115.7 Default

none

3.116 compute smd/tlsph/stress command

3.116.1 Syntax

```
compute ID group-ID smd/tlsph/stress
```

- ID, group-ID are documented in *compute* command
- smd/tlsph/stress = style name of this compute command

3.116.2 Examples

```
compute 1 all smd/tlsph/stress
```

3.116.3 Description

Define a computation that outputs the Cauchy stress tensor for particles interacting via the Total-Lagrangian SPH pair style.

See this PDF guide to using Smooth Mach Dynamics in LAMMPS.

3.116.4 Output info

This compute calculates a per-particle vector of vectors (tensors), which can be accessed by any command that uses per-particle values from a compute as input. See the *Howto output* doc page for an overview of LAMMPS output options.

The values will be given in *units* of pressure.

The per-particle vector has 7 entries. The first six entries correspond to the xx, yy, zz, xy, xz and yz components of the symmetric Cauchy stress tensor. The seventh entry is the second invariant of the stress tensor, i.e., the von Mises equivalent stress.

3.116.5 Restrictions

This compute 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.

This quantity will be computed only for particles which interact with the Total-Lagrangian SPH pair style.

3.116.6 Related commands

compute smd/tlsph/strain, *cmopute smd/tlsph/strain/rate*

3.116.7 Default

none

3.117 compute smd/triangle/vertices command

3.117.1 Syntax

```
compute ID group-ID smd/triangle/vertices
```

- ID, group-ID are documented in *compute* command
- smd/triangle/vertices = style name of this compute command

3.117.2 Examples

```
compute 1 all smd/triangle/vertices
```

3.117.3 Description

Define a computation that returns the coordinates of the vertices corresponding to the triangle-elements of a mesh created by the *fix smd/wall_surface*.

See this PDF guide to using Smooth Mach Dynamics in LAMMPS.

3.117.4 Output info

This compute returns a per-particle vector of vectors, which can be accessed by any command that uses per-particle values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

The per-particle vector has nine entries, (x1/y1/z1), (x2/y2/z2), and (x3/y3/z3) corresponding to the first, second, and third vertex of each triangle.

It is only meaningful to use this compute for a group of particles which is created via the *fix smd/wall_surface* command.

The output of this compute can be used with the dump2vtk_tris tool to generate a VTK representation of the smd/wall_surface mesh for visualization purposes.

The values will be given in *units* of distance.

3.117.5 Restrictions

This compute 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.

3.117.6 Related commands

fix smd/move/tri/surf, *fix smd/wall_surface*

3.117.7 Default

none

3.118 compute smd/ulsph/effm command

3.118.1 Syntax

```
compute ID group-ID smd/ulsph/effm
```

- ID, group-ID are documented in *compute* command
- smd/ulsph/effm = style name of this compute command

3.118.2 Examples

```
compute 1 all smd/ulsph/effm
```

3.118.3 Description

Define a computation that outputs the effective shear modulus for particles interacting via the updated Lagrangian SPH pair style.

See this PDF guide to using Smooth Mach Dynamics in LAMMPS.

3.118.4 Output info

This compute calculates a per-particle vector, which can be accessed by any command that uses per-particle values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

The per-particle vector contains the current effective per atom shear modulus as computed by the *pair smd/ulsph* pair style.

3.118.5 Restrictions

This compute 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. This compute can only be used for particles which interact with the updated Lagrangian SPH pair style.

3.118.6 Related commands

pair smd/ulsph

3.118.7 Default

none

3.119 compute smd/ulsph/num/neighs command

3.119.1 Syntax

```
compute ID group-ID smd/ulsph/num/neighs
```

- ID, group-ID are documented in *compute* command
- smd/ulsph/num/neighs = style name of this compute command

3.119.2 Examples

```
compute 1 all smd/ulsph/num/neighs
```

3.119.3 Description

Define a computation that returns the number of neighbor particles inside of the smoothing kernel radius for particles interacting via the updated Lagrangian SPH pair style.

See this PDF guide to using Smooth Mach Dynamics in LAMMPS.

3.119.4 Output info

This compute returns a per-particle vector, which can be accessed by any command that uses per-particle values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

The per-particle values will be given dimensionless, see *units*.

3.119.5 Restrictions

This compute 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. This compute can only be used for particles which interact with the updated Lagrangian SPH pair style.

3.119.6 Related commands

compute smd/tlsph/num/neighs

3.119.7 Default

none

3.120 compute smd/ulsph/strain command

3.120.1 Syntax

```
compute ID group-ID smd/ulsph/strain
```

- ID, group-ID are documented in *compute* command
- smd/ulsph/strain = style name of this compute command

3.120.2 Examples

```
compute 1 all smd/ulsph/strain
```

3.120.3 Description

Define a computation that outputs the logarithmic strain tensor. for particles interacting via the updated Lagrangian SPH pair style.

See this PDF guide to using Smooth Mach Dynamics in LAMMPS.

3.120.4 Output info

This compute calculates a per-particle tensor, which can be accessed by any command that uses per-particle values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

The per-particle vector has 6 entries, corresponding to the xx, yy, zz, xy, xz, yz components of the symmetric strain rate tensor.

The per-particle tensor values will be given dimensionless, see *units*.

3.120.5 Restrictions

This compute 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. This compute can only be used for particles which interact with the updated Lagrangian SPH pair style.

3.120.6 Related commands

compute smd/tlsph/strain

3.120.7 Default

none

3.121 compute smd/ulsph/strain/rate command

3.121.1 Syntax

```
compute ID group-ID smd/ulsph/strain/rate
```

- ID, group-ID are documented in *compute* command
- smd/ulsph/strain/rate = style name of this compute command

3.121.2 Examples

```
compute 1 all smd/ulsph/strain/rate
```

3.121.3 Description

Define a computation that outputs the rate of the logarithmic strain tensor for particles interacting via the updated Lagrangian SPH pair style.

See this PDF guide to using Smooth Mach Dynamics in LAMMPS.

3.121.4 Output info

This compute calculates a per-particle vector of vectors (tensors), which can be accessed by any command that uses per-particle values from a compute as input. See the *Howto output* doc page for an overview of LAMMPS output options.

The values will be given in *units* of one over time.

The per-particle vector has 6 entries, corresponding to the xx, yy, zz, xy, xz, yz components of the symmetric strain rate tensor.

3.121.5 Restrictions

This compute 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.

This compute can only be used for particles which interact with the updated Lagrangian SPH pair style.

3.121.6 Related commands

compute smd/tlsph/strain/rate

3.121.7 Default

none

3.122 compute smd/ulsph/stress command

3.122.1 Syntax

```
compute ID group-ID smd/ulsph/stress
```

- ID, group-ID are documented in *compute* command
- smd/ulsph/stress = style name of this compute command

3.122.2 Examples

```
compute 1 all smd/ulsph/stress
```

3.122.3 Description

Define a computation that outputs the Cauchy stress tensor.

See this PDF guide to using Smooth Mach Dynamics in LAMMPS.

3.122.4 Output info

This compute calculates a per-particle vector of vectors (tensors), which can be accessed by any command that uses per-particle values from a compute as input. See the *Howto output* doc page for an overview of LAMMPS output options.

The values will be given in *units* of pressure.

The per-particle vector has 7 entries. The first six entries correspond to the xx, yy, zz, xy, xz, yz components of the symmetric Cauchy stress tensor. The seventh entry is the second invariant of the stress tensor, i.e., the von Mises equivalent stress.

3.122.5 Restrictions

This compute 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. This compute can only be used for particles which interact with the updated Lagrangian SPH pair style.

3.122.6 Related commands

compute smd/ulsph/strain, *compute smd/ulsph/strain/rate compute smd/tlsph/stress*

3.122.7 Default

none

3.123 compute smd/vol command

3.123.1 Syntax

```
compute ID group-ID smd/vol
```

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

3.123.2 Examples

```
compute 1 all smd/vol
```

3.123.3 Description

Define a computation that provides the per-particle volume and the sum of the per-particle volumes of the group for which the compute is defined.

See this PDF guide to using Smooth Mach Dynamics in LAMMPS.

3.123.4 Output info

This compute calculates a per-particle vector, which can be accessed by any command that uses per-particle values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

The per-particle vector values will be given in *units* of volume.

Additionally, the compute returns a scalar, which is the sum of the per-particle volumes of the group for which the compute is defined.

3.123.5 Restrictions

This compute 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.

3.123.6 Related commands

compute smd/rho

3.123.7 Default

none

3.124 compute sna/atom command

3.125 compute snad/atom command

3.126 compute snav/atom command

3.127 compute snap command

3.128 compute sna/grid command

3.129 compute sna/grid/local command

3.129.1 Syntax

```
compute ID group-ID sna/atom rcutfac rfac0 twojmax R_1 R_2 ... w_1 w_2 ... keyword values ...
compute ID group-ID snad/atom rcutfac rfac0 twojmax R_1 R_2 ... w_1 w_2 ... keyword values ...
compute ID group-ID snav/atom rcutfac rfac0 twojmax R_1 R_2 ... w_1 w_2 ... keyword values ...
compute ID group-ID snap rcutfac rfac0 twojmax R_1 R_2 ... w_1 w_2 ... keyword values ...
compute ID group-ID snap rcutfac rfac0 twojmax R_1 R_2 ... w_1 w_2 ... keyword values ...
compute ID group-ID sna/grid nx ny nz rcutfac rfac0 twojmax R_1 R_2 ... w_1 w_2 ... keyword values .
↪..
compute ID group-ID sna/grid/local nx ny nz rcutfac rfac0 twojmax R_1 R_2 ... w_1 w_2 ... keyword␣
↪values ...
```

- ID, group-ID are documented in *compute* command
- sna/atom = style name of this compute command
- rcutfac = scale factor applied to all cutoff radii (positive real)
- rfac0 = parameter in distance to angle conversion (0 < rcutfac < 1)
- twojmax = band limit for bispectrum components (non-negative integer)
- R_1, R_2,... = list of cutoff radii, one for each type (distance units)
- w_1, w_2,... = list of neighbor weights, one for each type
- nx, ny, nz = number of grid points in x, y, and z directions (positive integer)
- zero or more keyword/value pairs may be appended
- keyword = *rmin0* or *switchflag* or *bzeroflag* or *quadraticflag* or *chem* or *bnormflag* or *wselfallflag* or *bikflag* or *switchinnerflag* or *sinner* or *dinner* or *dgradflag* or *nnn* or *wmode* or *delta*

  rmin0 value = parameter in distance to angle conversion (distance units)
  switchflag value = 0 or 1
    0 = do not use switching function
    1 = use switching function
  bzeroflag value = 0 or 1
    0 = do not subtract B0
    1 = subtract B0
  quadraticflag value = 0 or 1
    0 = do not generate quadratic terms

1 = generate quadratic terms
chem values = nelements elementlist
nelements = number of SNAP elements
elementlist = ntypes integers in range [0, nelements)
bnormflag value = 0 or 1
0 = do not normalize
1 = normalize bispectrum components
wselfallflag value = 0 or 1
0 = self-contribution only for element of central atom
1 = self-contribution for all elements
switchinnerflag value = 0 or 1
0 = do not use inner switching function
1 = use inner switching function
sinner values = sinnerlist
sinnerlist = ntypes values of Sinner (distance units)
dinner values = dinnerlist
dinnerlist = ntypes values of Dinner (distance units)
bikflag value = 0 or 1 (only implemented for compute snap)
0 = descriptors are summed over atoms of each type
1 = descriptors are listed separately for each atom
dgradflag value = 0 or 1 (only implemented for compute snap)
0 = descriptor gradients are summed over atoms of each type
1 = descriptor gradients are listed separately for each atom pair

- additional keyword = *nnn* or *wmode* or *delta*

nnn value = number of considered nearest neighbors to compute the bispectrum over a target specific number of neighbors (only implemented for compute sna/atom)
wmode value = weight function for finding optimal cutoff to match the target number of neighbors (required if nnn used, only implemented for compute sna/atom)
0 = heavyside weight function
1 = hyperbolic tangent weight function
delta value = transition interval centered at cutoff distance for hyperbolic tangent weight function (ignored if wmode=0, required if wmode=1, only implemented for compute sna/atom)

3.129.2 Examples

```
compute b all sna/atom 1.4 0.99363 6 2.0 2.4 0.75 1.0 rmin0 0.0
compute db all sna/atom 1.4 0.95 6 2.0 1.0
compute vb all sna/atom 1.4 0.95 6 2.0 1.0
compute snap all snap 1.4 0.95 6 2.0 1.0
compute snap all snap 1.0 0.99363 6 3.81 3.83 1.0 0.93 chem 2 0 1
compute snap all snap 1.0 0.99363 6 3.81 3.83 1.0 0.93 switchinnerflag 1 sinner 1.35 1.6 dinner 0.25 0.3
compute bgrid all sna/grid/local 200 200 200 1.4 0.95 6 2.0 1.0
compute bnnn all sna/atom 9.0 0.99363 8 0.5 1.0 rmin0 0.0 nnn 24 wmode 1 delta 0.2
```

3.129.3 Description

Define a computation that calculates a set of quantities related to the bispectrum components of the atoms in a group. These computes are used primarily for calculating the dependence of energy, force, and stress components on the linear coefficients in the *snap pair_style*, which is useful when training a SNAP potential to match target data.

Bispectrum components of an atom are order parameters characterizing the radial and angular distribution of neighbor atoms. The detailed mathematical definition is given in the paper by Thompson et al. *(Thompson)*

The position of a neighbor atom *i'* relative to a central atom *i* is a point within the 3D ball of radius $R_{ii'}$ = *rcutfac* $(R_i + R'_i)$

Bartok et al. *(Bartok)*, proposed mapping this 3D ball onto the 3-sphere, the surface of the unit ball in a four-dimensional space. The radial distance *r* within *R_ii'* is mapped on to a third polar angle θ_0 defined by,

$$\theta_0 = \mathrm{rfac0}\frac{r - r_{min0}}{R_{ii'} - r_{min0}}\pi$$

In this way, all possible neighbor positions are mapped on to a subset of the 3-sphere. Points south of the latitude θ_0 = *rfac0* π are excluded.

The natural basis for functions on the 3-sphere is formed by the representatives of *SU(2)*, the matrices $U^j_{m,m'}(\theta, \phi, \theta_0)$. These functions are better known as $D^j_{m,m'}$, the elements of the Wigner *D*-matrices *(Meremianin, Varshalovich, Mason)* The density of neighbors on the 3-sphere can be written as a sum of Dirac-delta functions, one for each neighbor, weighted by species and radial distance. Expanding this density function as a generalized Fourier series in the basis functions, we can write each Fourier coefficient as

$$u^j_{m,m'} = U^j_{m,m'}(0,0,0) + \sum_{r_{ii'}<R_{ii'}} f_c(r_{ii'}) w_{\mu_{i'}} U^j_{m,m'}(\theta_0, \theta, \phi)$$

The $w_{\mu_{i'}}$ neighbor weights are dimensionless numbers that depend on $\mu_{i'}$, the SNAP element of atom *i'*, while the central atom is arbitrarily assigned a unit weight. The function $f_c(r)$ ensures that the contribution of each neighbor atom goes smoothly to zero at $R_{ii'}$:

$$\begin{aligned} f_c(r) =& \frac{1}{2}(\cos(\pi\frac{r - r_{min0}}{R_{ii'} - r_{min0}}) + 1), r \le R_{ii'} \\ =& 0, r > R_{ii'} \end{aligned}$$

The expansion coefficients $u^j_{m,m'}$ are complex-valued and they are not directly useful as descriptors, because they are not invariant under rotation of the polar coordinate frame. However, the following scalar triple products of expansion coefficients can be shown to be real-valued and invariant under rotation *(Bartok)*.

$$B_{j_1,j_2,j} = \sum_{m_1,m'_1=-j_1}^{j_1} \sum_{m_2,m'_2=-j_2}^{j_2} \sum_{m,m'=-j}^{j} (u^j_{m,m'})^* H \begin{matrix} jmm' \\ j_1 m_1 m'_1 \\ j_2 m_2 m'_2 \end{matrix} u^{j_1}_{m_1,m'_1} u^{j_2}_{m_2,m'_2}$$

The constants $H^{jmm'}_{j_1 m_1 m_{1'}, j_2 m_2 m_{2'}}$ are coupling coefficients, analogous to Clebsch-Gordan coefficients for rotations on the 2-sphere. These invariants are the components of the bispectrum and these are the quantities calculated by the compute *sna/atom*. They characterize the strength of density correlations at three points on the 3-sphere. The j2=0 subset form the power spectrum, which characterizes the correlations of two points. The lowest-order components describe the coarsest features of the density function, while higher-order components reflect finer detail. Each bispectrum component contains terms that depend on the positions of up to 4 atoms (3 neighbors and the central atom).

Compute *snad/atom* calculates the derivative of the bispectrum components summed separately for each LAMMPS atom type:

$$-\sum_{i' \in I} \frac{\partial B^{i'}_{j_1,j_2,j}}{\partial \mathbf{r}_i}$$

The sum is over all atoms *i'* of atom type *I*. For each atom *i*, this compute evaluates the above expression for each direction, each atom type, and each bispectrum component. See section below on output for a detailed explanation.

Compute *snav/atom* calculates the virial contribution due to the derivatives:

$$-\mathbf{r}_i \otimes \sum_{i' \in I} \frac{\partial B^{i'}_{j_1,j_2,j}}{\partial \mathbf{r}_i}$$

Again, the sum is over all atoms *i'* of atom type *I*. For each atom *i*, this compute evaluates the above expression for each of the six virial components, each atom type, and each bispectrum component. See section below on output for a detailed explanation.

Compute *snap* calculates a global array containing information related to all three of the above per-atom computes *sna/atom*, *snad/atom*, and *snav/atom*. The first row of the array contains the summation of *sna/atom* over all atoms, but broken out by type. The last six rows of the array contain the summation of *snav/atom* over all atoms, broken out by type. In between these are 3*N rows containing the same values computed by *snad/atom* (these are already summed over all atoms and broken out by type). The element in the last column of each row contains the potential energy, force, or stress, according to the row. These quantities correspond to the user-specified reference potential that must be subtracted from the target data when fitting SNAP. The potential energy calculation uses the built in compute *thermo_pe*. The stress calculation uses a compute called *snap_press* that is automatically created behind the scenes, according to the following command:

```
compute snap_press all pressure NULL virial
```

See section below on output for a detailed explanation of the data layout in the global array.

Added in version 3Aug2022.

The compute *sna/grid* and *sna/grid/local* commands calculate bispectrum components for a regular grid of points. These are calculated from the local density of nearby atoms *i'* around each grid point, as if there was a central atom *i* at the grid point. This is useful for characterizing fine-scale structure in a configuration of atoms, and it is used in the MALA package to build machine-learning surrogates for finite-temperature Kohn-Sham density functional theory (*Ellis et al.*) Neighbor atoms not in the group do not contribute to the bispectrum components of the grid points. The distance cutoff $R_{ii'}$ assumes that *i* has the same type as the neighbor atom *i'*.

Compute *sna/grid* calculates a global array containing bispectrum components for a regular grid of points. The grid is aligned with the current box dimensions, with the first point at the box origin, and forming a regular 3d array with *nx*, *ny*, and *nz* points in the x, y, and z directions. For triclinic boxes, the array is congruent with the periodic lattice vectors a, b, and c. The array contains one row for each of the $nx \times ny \times nz$ grid points, looping over the index for *ix* fastest, then *iy*, and *iz* slowest. Each row of the array contains the *x*, *y*, and *z* coordinates of the grid point, followed by the bispectrum components. See section below on output for a detailed explanation of the data layout in the global array.

Compute *sna/grid/local* calculates bispectrum components of a regular grid of points similarly to compute *sna/grid* described above. However, because the array is local, it contains only rows for grid points that are local to the processor subdomain. The global grid of $nx \times ny \times nz$ points is still laid out in space the same as for *sna/grid*, but grid points are strictly partitioned, so that every grid point appears in one and only one local array. The array contains one row for each of the local grid points, looping over the global index *ix* fastest, then *iy*, and *iz* slowest. Each row of the array contains the global indexes *ix*, *iy*, and *iz* first, followed by the *x*, *y*, and *z* coordinates of the grid point, followed by the bispectrum components. See section below on output for a detailed explanation of the data layout in the global array.

The value of all bispectrum components will be zero for atoms not in the group. Neighbor atoms not in the group do not contribute to the bispectrum of atoms in the group.

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

The argument *rcutfac* is a scale factor that controls the ratio of atomic radius to radial cutoff distance.

The argument *rfac0* and the optional keyword *rmin0* define the linear mapping from radial distance to polar angle $theta_0$ on the 3-sphere, given above.

The argument *twojmax* defines which bispectrum components are generated. See section below on output for a detailed explanation of the number of bispectrum components and the ordered in which they are listed.

The keyword *switchflag* can be used to turn off the switching function $f_c(r)$.

The keyword *bzeroflag* determines whether or not *B0*, the bispectrum components of an atom with no neighbors, are subtracted from the calculated bispectrum components. This optional keyword normally only affects compute *sna/atom*. However, when *quadraticflag* is on, it also affects *snad/atom* and *snav/atom*.

The keyword *quadraticflag* determines whether or not the quadratic combinations of bispectrum quantities are generated. These are formed by taking the outer product of the vector of bispectrum components with itself. See section below on output for a detailed explanation of the number of quadratic terms and the ordered in which they are listed.

The keyword *chem* activates the explicit multi-element variant of the SNAP bispectrum components. The argument *nelements* specifies the number of SNAP elements that will be handled. This is followed by *elementlist*, a list of integers of length *ntypes*, with values in the range [0, *nelements*), which maps each LAMMPS type to one of the SNAP elements. Note that multiple LAMMPS types can be mapped to the same element, and some elements may be mapped by no LAMMPS type. However, in typical use cases (training SNAP potentials) the mapping from LAMMPS types to elements is one-to-one.

The explicit multi-element variant invoked by the *chem* keyword partitions the density of neighbors into partial densities for each chemical element. This is described in detail in the paper by *Cusentino et al.* The bispectrum components are indexed on ordered triplets of elements:

$$B^{\kappa\lambda\mu}_{j_1,j_2,j} = \sum_{m_1,m_1'=-j_1}^{j_1} \sum_{m_2,m_2'=-j_2}^{j_2} \sum_{m,m'=-j}^{j} (u^{\mu}_{j,m,m'})^* H \begin{matrix} jmm' \\ j_1 m_1 m_1' \\ j_2 m_2 m_2' \end{matrix} u^{\kappa}_{j_1,m_1,m_1'} u^{\lambda}_{j_2,m_2,m_2'}$$

where $u^{\mu}_{j,m,m'}$ is an expansion coefficient for the partial density of neighbors of element μ

$$u^{\mu}_{j,m,m'} = w^{self}_{\mu_i\mu} U^{j,m,m'}(0,0,0) + \sum_{r_{ii'}<R_{ii'}} \delta_{\mu\mu_{i'}} f_c(r_{ii'}) w_{\mu_{i'}} U^{j,m,m'}(\theta_0,\theta,\phi)$$

where $w^{self}_{\mu_i\mu}$ is the self-contribution, which is either 1 or 0 (see keyword *wselfallflag* below), $\delta_{\mu\mu_{i'}}$ indicates that the sum is only over neighbor atoms of element μ, and all other quantities are the same as those appearing in the original equation for $u^{j}_{m,m'}$ given above.

The keyword *wselfallflag* defines the rule used for the self-contribution. If *wselfallflag* is on, then $w^{self}_{\mu_i\mu} = 1$. If it is off then $w^{self}_{\mu_i\mu} = 0$, except in the case of $\mu_i = \mu$, when $w^{self}_{\mu_i\mu} = 1$. When the *chem* keyword is not used, this keyword has no effect.

The keyword *bnormflag* determines whether or not the bispectrum component $B_{j_1,j_2,j}$ is divided by a factor of $2j+1$. This normalization simplifies force calculations because of the following symmetry relation

$$\frac{B_{j_1,j_2,j}}{2j+1} = \frac{B_{j,j_2,j_1}}{2j_1+1} = \frac{B_{j_1,j,j_2}}{2j_2+1}$$

This option is typically used in conjunction with the *chem* keyword, and LAMMPS will generate a warning if both *chem* and *bnormflag* are not both set or not both unset.

The keyword *switchinnerflag* with value 1 activates an additional radial switching function similar to $f_c(r)$ above, but acting to switch off smoothly contributions from neighbor atoms at short separation distances. This is useful when SNAP is used in combination with a simple repulsive potential. For a neighbor atom at distance r, its contribution is scaled by a multiplicative factor $f_{inner}(r)$ defined as follows:

$$\begin{aligned} &=0, r \le S_{inner} - D_{inner} \\ f_{inner}(r) &=\frac{1}{2}(1-\cos(\frac{\pi}{2}(1+\frac{r-S_{inner}}{D_{inner}})), S_{inner}-D_{inner} < r \le S_{inner}+D_{inner} \\ &=1, r > S_{inner}+D_{inner} \end{aligned}$$

where the switching region is centered at S_{inner} and it extends a distance D_{inner} to the left and to the right of this. With this option, additional keywords *sinner* and *dinner* must be used, each followed by *ntypes* values for S_{inner} and D_{inner}, respectively. When the central atom and the neighbor atom have different types, the values of S_{inner} and D_{inner} are the arithmetic means of the values for both types.

The keywords *bikflag* and *dgradflag* are only used by compute *snap*. The keyword *bikflag* determines whether or not to list the descriptors of each atom separately, or sum them together and list in a single row. If *bikflag* is set to *0* then a single bispectrum row is used, which contains the per-atom bispectrum descriptors $B_{i,k}$ summed over all atoms i to produce B_k. If *bikflag* is set to *1* this is replaced by a separate per-atom bispectrum row for each atom. In this case, the entries in the final column for these rows are set to zero.

The keyword *dgradflag* determines whether to sum atom gradients or list them separately. If *dgradflag* is set to 0, the bispectrum descriptor gradients w.r.t. atom j are summed over all atoms i' of type I (similar to *snad/atom* above). If *dgradflag* is set to 1, gradients are listed separately for each pair of atoms. Each row corresponds to a single term $\frac{\partial B_{i,k}}{\partial r_j^a}$ where r_j^a is the *a-th* position coordinate of the atom with global index j. This also changes the number of columns to be equal to the number of bispectrum components, with 3 additional columns representing the indices i, j, and a, as explained more in the Output info section below. The option *dgradflag=1* requires that *bikflag=1*.

Note

Using *dgradflag* = 1 produces a global array with $N + 3N^2 + 1$ rows which becomes expensive for systems with more than 1000 atoms.

Note

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

The keyword *nnn* allows for the calculation of the bispectrum over a specific target number of neighbors. This option is only implemented for the compute *sna/atom*. An optimal cutoff radius for defining the neighborhood of the central atom is calculated by means of a dichotomy algorithm. This iterative process allows to assign weights to neighboring atoms in order to match the total sum of weights with the target number of neighbors. Depending on the radial weight function used in that process, the cutoff radius can fluctuate a lot in the presence of thermal noise. Therefore, in addition to the *nnn* keyword, the keyword *wmode* allows to choose whether a Heaviside (*wmode* = 0) function or a Hyperbolic tangent function (*wmode* = 1) should be used. If the Heaviside function is used, the cutoff radius exactly matches the distance between the central atom an its *nnn*'th neighbor. However, in the case of the hyperbolic tangent function, the dichotomy algorithm allows to span the weights over a distance *delta* in order to reduce fluctuations in the resulting local atomic environment fingerprint. The detailed formalism is given in the paper by Lafourcade et al. *(Lafourcade)*.

3.129.4 Output info

Compute *sna/atom* calculates a per-atom array, each column corresponding to a particular bispectrum component. The total number of columns and the identity of the bispectrum component contained in each column depend of the value of *twojmax*, as described by the following piece of python code:

```
for j1 in range(0,twojmax+1):
   for j2 in range(0,j1+1):
      for j in range(j1-j2,min(twojmax,j1+j2)+1,2):
         if (j>=j1): print j1/2.,j2/2.,j/2.
```

There are $m(m+1)/2$ descriptors with last index j, where $m = \lfloor j \rfloor + 1$. Hence, for even *twojmax* = 2(m-1), $K = m(m+1)(2m+1)/6$, the m-th pyramidal number, and for odd *twojmax* = 2 m-1, $K = m(m+1)(m+2)/3$, twice the m-th tetrahedral number.

Note

the *diagonal* keyword allowing other possible choices for the number of bispectrum components was removed in 2019, since all potentials use the value of 3, corresponding to the above set of bispectrum components.

Compute *snad/atom* evaluates a per-atom array. The columns are arranged into *ntypes* blocks, listed in order of atom type I. Each block contains three sub-blocks corresponding to the x, y, and z components of the atom position. Each of these sub-blocks contains K columns for the K bispectrum components, the same as for compute *sna/atom*

Compute *snav/atom* evaluates a per-atom array. The columns are arranged into *ntypes* blocks, listed in order of atom type I. Each block contains six sub-blocks corresponding to the xx, yy, zz, yz, xz, and xy components of the virial tensor in Voigt notation. Each of these sub-blocks contains K columns for the K bispectrum components, the same as for compute *sna/atom*

Compute *snap* evaluates a global array. The columns are arranged into *ntypes* blocks, listed in order of atom type I. Each block contains one column for each bispectrum component, the same as for compute *sna/atom*. A final column contains the corresponding energy, force component on an atom, or virial stress component. The rows of the array appear in the following order:

- 1 row: *sna/atom* quantities summed for all atoms of type I
- 3*N rows: *snad/atom* quantities, with derivatives w.r.t. x, y, and z coordinate of atom i appearing in consecutive rows. The atoms are sorted based on atom ID.
- 6 rows: *snav/atom* quantities summed for all atoms of type I

For example, if K =30 and ntypes=1, the number of columns in the per-atom arrays generated by *sna/atom*, *snad/atom*, and *snav/atom* are 30, 90, and 180, respectively. With *quadratic* value=1, the numbers of columns are 930, 2790, and 5580, respectively. The number of columns in the global array generated by *snap* are 31, and 931, respectively, while the number of rows is 1+3*N+6, where N is the total number of atoms.

Compute *sna/grid* evaluates a global array. The array contains one row for each of the $nx \times ny \times nz$ grid points, looping over the index for ix fastest, then iy, and iz slowest. Each row of the array contains the x, y, and z coordinates of the grid point, followed by the bispectrum components.

Compute *sna/grid/local* evaluates a local array. The array contains one row for each of the local grid points, looping over the global index ix fastest, then iy, and iz slowest. Each row of the array contains the global indexes ix, iy, and iz first, followed by the x, y, and z coordinates of the grid point, followed by the bispectrum components.

If the *quadratic* keyword value is set to 1, then additional columns are generated, corresponding to the products of all distinct pairs of bispectrum components. If the number of bispectrum components is K, then the number of distinct pairs is $K(K+1)/2$. For compute *sna/atom* these columns are appended to existing K columns. The ordering of quadratic terms is upper-triangular, (1,1),(1,2)...(1,K),(2,1)...(K-1,K-1),(K-1,K),(K,K). For computes *snad/atom* and *snav/atom*

each set of $K(K+1)/2$ additional columns is inserted directly after each of sub-block of linear terms i.e. linear and quadratic terms are contiguous. So the nesting order from inside to outside is bispectrum component, linear then quadratic, vector/tensor component, type.

If the *chem* keyword is used, then the data is arranged into N_{elem}^3 sub-blocks, each sub-block corresponding to a particular chemical labeling $\kappa\lambda\mu$ with the last label changing fastest. Each sub-block contains K bispectrum components. For the purposes of handling contributions to force, virial, and quadratic combinations, these N_{elem}^3 sub-blocks are treated as a single block of KN_{elem}^3 columns.

If the *bik* keyword is set to 1, the structure of the snap array is expanded. The first N rows of the snap array correspond to $B_{i,k}$ instead of a single row summed over atoms i. In this case, the entries in the final column for these rows are set to zero. Also, each row contains only non-zero entries for the columns corresponding to the type of that atom. This is not true in the case of *dgradflag* keyword = 1 (see below).

If the *dgradflag* keyword is set to 1, this changes the structure of the global array completely. Here the *snad/atom* quantities are replaced with rows corresponding to descriptor gradient components on single atoms:

$$\frac{\partial B_{i,k}}{\partial r_j^a}$$

where r_j^a is the *a-th* position coordinate of the atom with global index j. The rows are organized in chunks, where each chunk corresponds to an atom with global index j. The rows in an atom j chunk correspond to atoms with global index i. The total number of rows for these descriptor gradients is therefore $3N^2$. The number of columns is equal to the number of bispectrum components, plus 3 additional left-most columns representing the global atom indices i, j, and Cartesian direction a (0, 1, 2, for x, y, z). The first 3 columns of the first N rows belong to the reference potential force components. The remaining K columns contain the $B_{i,k}$ per-atom descriptors corresponding to the non-zero entries obtained when *bikflag* = 1. The first column of the last row, after the first $N+3N^2$ rows, contains the reference potential energy. The virial components are not used with this option. The total number of rows is therefore $N+3N^2+1$ and the number of columns is $K+3$.

These values can be accessed by any command that uses per-atom values from a compute as input. See the *Howto output* doc page for an overview of LAMMPS output options. To see how this command can be used within a Python workflow to train SNAP potentials, see the examples in FitSNAP.

3.129.5 Restrictions

These computes are part of the ML-SNAP package. They are only enabled if LAMMPS was built with that package. See the *Build package* page for more info.

3.129.6 Related commands

pair_style snap compute slcsa/atom

3.129.7 Default

The optional keyword defaults are *rmin0* = 0, *switchflag* = 1, *bzeroflag* = 1, *quadraticflag* = 0, *bnormflag* = 0, *wselfallflag* = 0, *switchinnerflag* = 0, *nnn* = -1, *wmode* = 0, *delta* = 1.e-3

(Thompson) Thompson, Swiler, Trott, Foiles, Tucker, J Comp Phys, 285, 316, (2015).

(Bartok) Bartok, Payne, Risi, Csanyi, Phys Rev Lett, 104, 136403 (2010).

(Meremianin) Meremianin, J. Phys. A, 39, 3099 (2006).

(Varshalovich) Varshalovich, Moskalev, Khersonskii, Quantum Theory of Angular Momentum, World Scientific, Singapore (1987).

(Mason) J. K. Mason, Acta Cryst A65, 259 (2009).

(Cusentino) Cusentino, Wood, Thompson, J Phys Chem A, 124, 5456, (2020)

(Ellis) Ellis, Fiedler, Popoola, Modine, Stephens, Thompson, Cangi, Rajamanickam, Phys Rev B, 104, 035120, (2021)

(Lafourcade) Lafourcade, Maillet, Denoual, Duval, Allera, Goryaeva, and Marinica, Comp. Mat. Science, 230, 112534 (2023)

3.130 compute sph/e/atom command

3.130.1 Syntax

```
compute ID group-ID sph/e/atom
```

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

3.130.2 Examples

```
compute 1 all sph/e/atom
```

3.130.3 Description

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

The internal energy is the energy associated with the internal degrees of freedom of an SPH particle, i.e. a Smooth-Particle Hydrodynamics particle.

See this PDF guide to using SPH in LAMMPS.

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

3.130.4 Output info

This compute calculates a per-atom vector, which can be accessed by any command that uses per-atom values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

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

3.130.5 Restrictions

This compute 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.

3.130.6 Related commands

dump custom

3.130.7 Default

none

3.131 compute sph/rho/atom command

3.131.1 Syntax

```
compute ID group-ID sph/rho/atom
```

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

3.131.2 Examples

```
compute 1 all sph/rho/atom
```

3.131.3 Description

Define a computation that calculates the per-atom SPH density for each atom in a group, i.e. a Smooth-Particle Hydrodynamics density.

The SPH density is the mass density of an SPH particle, calculated by kernel function interpolation using "pair style sph/rhosum".

See this PDF guide to using SPH in LAMMPS.

The value of the SPH density will be 0.0 for atoms not in the specified compute group.

3.131.4 Output info

This compute calculates a per-atom vector, which can be accessed by any command that uses per-atom values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

The per-atom vector values will be in mass/volume *units*.

3.131.5 Restrictions

This compute 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.

3.131.6 Related commands

dump custom

3.131.7 Default

none

3.132 compute sph/t/atom command

3.132.1 Syntax

```
compute ID group-ID sph/t/atom
```

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

3.132.2 Examples

```
compute 1 all sph/t/atom
```

3.132.3 Description

Define a computation that calculates the per-atom internal temperature for each atom in a group.

The internal temperature is the ratio of internal energy over the heat capacity associated with the internal degrees of freedom of an SPH particles, i.e. a Smooth-Particle Hydrodynamics particle.

$$T_{int} = E_{int}/C_{V,int}$$

See this PDF guide to using SPH in LAMMPS.

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

3.132.4 Output info

This compute calculates a per-atom vector, which can be accessed by any command that uses per-atom values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

The per-atom vector values will be in temperature *units*.

3.132.5 Restrictions

This compute 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.

3.132.6 Related commands

dump custom

3.132.7 Default

none

3.133 compute spin command

3.133.1 Syntax

```
compute ID group-ID spin
```

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

3.133.2 Examples

```
compute out_mag all spin
```

3.133.3 Description

Define a computation that calculates magnetic quantities for a system of atoms having spins.

This compute calculates the following 6 magnetic quantities:

- the three first quantities are the x,y and z coordinates of the total magnetization,
- the fourth quantity is the norm of the total magnetization,
- The fifth quantity is the magnetic energy (in eV),
- The sixth one is referred to as the spin temperature, according to the work of *(Nurdin)*.

The simplest way to output the results of the compute spin calculation is to define some of the quantities as variables, and to use the thermo and thermo_style commands, for example:

```
compute out_mag         all spin

variable mag_z          equal c_out_mag[3]
variable mag_norm       equal c_out_mag[4]
variable temp_mag       equal c_out_mag[6]

thermo                  10
thermo_style            custom step v_mag_z v_mag_norm v_temp_mag
```

This series of commands evaluates the total magnetization along z, the norm of the total magnetization, and the magnetic temperature. Three variables are assigned to those quantities. The thermo and thermo_style commands print them every 10 timesteps.

3.133.4 Output info

The array values are "intensive". The array values will be in metal units (*units*).

3.133.5 Restrictions

The *spin* compute is part of the SPIN package. This compute is only enabled if LAMMPS was built with this package. See the *Build package* page for more info. The atom_style has to be "spin" for this compute to be valid.

Related commands:

none

3.133.6 Default

none

(Nurdin) Nurdin and Schotte Phys Rev E, 61(4), 3579 (2000)

3.134 compute stress/atom command

3.135 compute centroid/stress/atom command

3.135.1 Syntax

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

- ID, group-ID are documented in *compute* command
- style = *stress/atom* or *centroid/stress/atom*
- temp-ID = ID of compute that calculates temperature, can be NULL if not needed
- zero or more keywords may be appended
- keyword = *ke* or *pair* or *bond* or *angle* or *dihedral* or *improper* or *kspace* or *fix* or *virial*

3.135.2 Examples

```
compute 1 mobile stress/atom NULL
compute 1 mobile stress/atom myRamp
compute 1 all stress/atom NULL pair bond
compute 1 all centroid/stress/atom NULL bond dihedral improper
```

3.135.3 Description

Define a computation that computes per-atom stress tensor for each atom in a group. In case of compute *stress/atom*, the tensor for each atom is symmetric with 6 components and is stored as a 6-element vector in the following order: *xx*, *yy*, *zz*, *xy*, *xz*, *yz*. In case of compute *centroid/stress/atom*, the tensor for each atom is asymmetric with 9 components and is stored as a 9-element vector in the following order: *xx*, *yy*, *zz*, *xy*, *xz*, *yz*, *yx*, *zx*, *zy*. See the *compute pressure* command if you want the stress tensor (pressure) of the entire system.

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

$$S_{ab} = -mv_a v_b - W_{ab}$$

The first term is a kinetic energy contribution for atom I. See details below on how the specified *temp-ID* can affect the velocities used in this calculation. The second term is the virial contribution due to intra and intermolecular interactions, where the exact computation details are determined by the compute style.

In case of compute *stress/atom*, the virial contribution is:

$$\begin{aligned} W_{ab} = & \frac{1}{2}\sum_{n=1}^{N_p}(r_{1_a}F_{1_b} + r_{2_a}F_{2_b}) + \frac{1}{2}\sum_{n=1}^{N_b}(r_{1_a}F_{1_b} + r_{2_a}F_{2_b}) \\ & + \frac{1}{3}\sum_{n=1}^{N_a}(r_{1_a}F_{1_b} + r_{2_a}F_{2_b} + r_{3_a}F_{3_b}) + \frac{1}{4}\sum_{n=1}^{N_d}(r_{1_a}F_{1_b} + r_{2_a}F_{2_b} + r_{3_a}F_{3_b} + r_{4_a}F_{4_b}) \\ & + \frac{1}{4}\sum_{n=1}^{N_i}(r_{1_a}F_{1_b} + r_{2_a}F_{2_b} + r_{3_a}F_{3_b} + r_{4_a}F_{4_b}) + \mathrm{Kspace}(r_{i_a}, F_{i_b}) + \sum_{n=1}^{N_f} r_{i_a}F_{i_b} \end{aligned}$$

The first term is a pairwise energy contribution where n loops over the N_p neighbors of atom I, $\mathbf{r}_1$ and $\mathbf{r}_2$ are the positions of the two atoms in the pairwise interaction, and $\mathbf{F}_1$ and $\mathbf{F}_2$ are the forces on the two atoms resulting from the pairwise interaction. The second term is a bond contribution of similar form for the N_b bonds which atom I is part of. There are similar terms for the N_a angle, N_d dihedral, and N_i improper interactions atom I is part of. There is also a term for the KSpace contribution from long-range Coulombic interactions, if defined. Finally, there is a term for the N_f *fixes* that apply internal constraint forces to atom I. Currently, only the *fix shake* and *fix rigid* commands contribute to this term. As the coefficients in the formula imply, a virial contribution produced by a small set of atoms (e.g. 4 atoms in a dihedral or 3 atoms in a Tersoff 3-body interaction) is assigned in equal portions to each atom in the set. E.g. 1/4 of the dihedral virial to each of the 4 atoms, or 1/3 of the fix virial due to SHAKE constraints applied to atoms in a water molecule via the *fix shake* command. As an exception, the virial contribution from constraint forces in *fix rigid* on each atom is computed from the constraint force acting on the corresponding atom and its position, i.e. the total virial is not equally distributed.

In case of compute *centroid/stress/atom*, the virial contribution is:

$$\begin{aligned} W_{ab} = & \sum_{n=1}^{N_p} r_{I0_a}F_{I_b} + \sum_{n=1}^{N_b} r_{I0_a}F_{I_b} + \sum_{n=1}^{N_a} r_{I0_a}F_{I_b} + \sum_{n=1}^{N_d} r_{I0_a}F_{I_b} + \sum_{n=1}^{N_i} r_{I0_a}F_{I_b} \\ & + \mathrm{Kspace}(r_{i_a}, F_{i_b}) + \sum_{n=1}^{N_f} r_{i_a}F_{i_b} \end{aligned}$$

As with compute *stress/atom*, the first, second, third, fourth and fifth terms are pairwise, bond, angle, dihedral and improper contributions, but instead of assigning the virial contribution equally to each atom, only the force $\mathbf{F}_I$ acting on atom I due to the interaction and the relative position $\mathbf{r}_{I0}$ of the atom I to the geometric center of the interacting atoms, i.e. centroid, is used. As the geometric center is different for each interaction, the $\mathbf{r}_{I0}$ also differs. The sixth term, Kspace contribution, is computed identically to compute *stress/atom*. The seventh term is handed differently depending on if the constraint forces are due to *fix shake* or *fix rigid*. In case of SHAKE constraints, each distance constraint is handed as a pairwise interaction. E.g. in case of a water molecule, two OH and one HH distance constraints are treated as three pairwise interactions. In case of *fix rigid*, all constraint forces in the molecule are treated as a single many-body interaction with a single centroid position. In case of water molecule, the formula expression would become identical to that of the three-body angle interaction. Although the total system virial is the same as compute *stress/atom*, compute *centroid/stress/atom* is know to result in more consistent heat flux values for angle, dihedrals, improper and constraint force contributions when computed via *compute heat/flux*.

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

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

Details of how compute *stress/atom* obtains the virial for individual atoms for either pairwise or many-body potentials, and including the effects of periodic boundary conditions is discussed in *(Thompson)*. The basic idea for many-body potentials is to treat each component of the force computation between a small cluster of atoms in the same manner as in the formula above for bond, angle, dihedral, etc interactions. Namely the quantity $\mathbf{r} \cdot \mathbf{F}$ is summed over the atoms in the interaction, with the r vectors unwrapped by periodic boundaries so that the cluster of atoms is close together. The total contribution for the cluster interaction is divided evenly among those atoms.

Details of how compute *centroid/stress/atom* obtains the virial for individual atoms are given in *(Surblys2019)* and *(Surblys2021)*, where the idea is that the virial of the atom I is the result of only the force $\mathbf{F}_I$ on the atom due to the interaction and its positional vector $\mathbf{r}_{I0}$, relative to the geometric center of the interacting atoms, regardless of the number of participating atoms. The periodic boundary treatment is identical to that of compute *stress/atom*, and both of them reduce to identical expressions for two-body interactions, i.e. computed values for contributions from bonds and two-body pair styles, such as *Lennard-Jones*, will be the same, while contributions from angles, dihedrals and impropers will be different.

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

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

The *temp-ID* argument can be used to affect the per-atom velocities used in the kinetic energy contribution to the total stress. If the kinetic energy is not included in the stress, than the temperature compute is not used and can be specified as NULL. If the kinetic energy is included and you wish to use atom velocities as-is, then *temp-ID* can also be specified as NULL. If desired, the specified temperature compute can be one that subtracts off a bias to leave each atom with only a thermal velocity to use in the formula above, e.g. by subtracting a background streaming velocity. See the doc pages for individual *compute commands* to determine which ones include a bias.

Note that as defined in the formula, per-atom stress is the negative of the per-atom pressure tensor. It is also really a stress*volume formulation, meaning the computed quantity is in units of pressure*volume. It would need to be divided by a per-atom volume to have units of stress (pressure), but an individual atom's volume is not well defined or easy to compute in a deformed solid or a liquid. See the *compute voronoi/atom* command for one possible way to estimate a per-atom volume.

Thus, if the diagonal components of the per-atom stress tensor are summed for all atoms in the system and the sum is divided by dV, where d = dimension and V is the volume of the system, the result should be $-P$, where P is the total pressure of the system.

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

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

Note

The per-atom stress does not include any Lennard-Jones tail corrections to the pressure added by the *pair_modify tail yes* command, since those are contributions to the global system pressure.

The compute stress/atom can be used in a number of ways. Here is an example to compute a 1-d pressure profile in x-direction across the complete simulation box. You will need to adjust the number of bins and the selections for time averaging to your specific simulation. This assumes that the dimensions of the simulation cell does not change.

```
# set number of bins
variable nbins index 20
variable fraction equal 1.0/v_nbins
# define bins as chunks
compute cchunk all chunk/atom bin/1d x lower ${fraction} units reduced
compute stress all stress/atom NULL
# apply conversion to pressure early since we have no variable style for processing chunks
variable press atom -(c_stress[1]+c_stress[2]+c_stress[3])/(3.0*vol*${fraction})
compute binpress all reduce/chunk cchunk sum v_press
fix avg all ave/time 10 40 400 c_binpress mode vector file ave_stress.txt
```

3.135.4 Output info

Compute *stress/atom* calculates a per-atom array with 6 columns, which can be accessed by indices 1-6 by any command that uses per-atom values from a compute as input. Compute *centroid/stress/atom* produces a per-atom array with 9 columns, but otherwise can be used in an identical manner to compute *stress/atom*. See the *Howto output* page for an overview of LAMMPS output options.

The ordering of the 6 columns for *stress/atom* is as follows: xx, yy, zz, xy, xz, yz. The ordering of the 9 columns for *centroid/stress/atom* is as follows: xx, yy, zz, xy, xz, yz, yx, zx, zy.

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

3.135.5 Restrictions

Currently, compute *centroid/stress/atom* does not support pair styles with many-body interactions (*EAM* is an exception, since its computations are performed pairwise), nor granular pair styles with pairwise forces which are not aligned with the vector between the pair of particles. All bond styles are supported. All angle, dihedral, improper styles are supported with the exception of INTEL and KOKKOS variants of specific styles. It also does not support models with long-range Coulombic or dispersion forces, i.e. the kspace_style command in LAMMPS. It also does not implement the following fixes which add rigid-body constraints: *fix rigid/** and the OpenMP accelerated version of *fix rigid/small*, while all other *fix rigid/*/small* are implemented.

LAMMPS will generate an error if one of these options is included in your model. Extension of centroid stress calculations to these force and fix styles is planned for the future.

3.135.6 Related commands

compute pe, *compute pressure*

3.135.7 Default

By default the compute includes contributions from the keywords: ke pair bond angle dihedral improper kspace fix

---

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

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

(Thompson) Thompson, Plimpton, Mattson, J Chem Phys, 131, 154107 (2009).

(Surblys2019) Surblys, Matsubara, Kikugawa, Ohara, Phys Rev E, 99, 051301(R) (2019).

(Surblys2021) Surblys, Matsubara, Kikugawa, Ohara, J Appl Phys 130, 215104 (2021).

3.136 compute stress/cartesian command

3.136.1 Syntax

```
compute ID group-ID stress/cartesian args
```

- ID, group-ID are documented in *compute* command
- args = argument specific to the compute style

stress/cartesian args = dim1 bin_width1 dim2 bin_width2 keyword
  dim1 = x or y or z
  bin_width1 = width of the bin
  dim2 = x or y or z or NULL
  bin_width2 = width of the bin
  keyword = ke or pair or bond

3.136.2 Examples

```
compute 1 all stress/cartesian x 0.1 NULL 0
compute 1 all stress/cartesian y 0.1 z 0.1
compute 1 all stress/cartesian x 0.1 NULL 0 ke pair
```

3.136.3 Description

Compute *stress/cartesian* defines computations that calculate profiles of the diagonal components of the local stress tensor over one or two Cartesian dimensions, as described in *(Ikeshoji)*. The stress tensor is split into a kinetic contribution P^k and a virial contribution P^v. The sum gives the total stress tensor $P = P^k + P^v$. This compute obeys momentum balance through fluid interfaces. They use the Irving–Kirkwood contour, which is the straight line between particle pairs.

Added in version 15Jun2023: Added support for bond styles

This compute only supports pair and bond (no angle, dihedral, improper, or kspace) forces. By default, if no extra keywords are specified, all supported contributions to the stress are included (ke, pair, bond). If any keywords are specified, then only those components are summed.

3.136.4 Output info

The output columns for *stress/cartesian* are the position of the center of the local volume in the first and second dimensions, number density, P^k_{xx}, P^k_{yy}, P^k_{zz}, P^v_{xx}, P^v_{yy}, and P^v_{zz}. There are 8 columns when one dimension is specified and 9 columns when two dimensions are specified. The number of bins (rows) is $(L_1/b_1)(L_2/b_2)$, where L_1 and L_2 are the lengths of the simulation box in the specified dimensions and b_1 and b_2 are the specified bin widths. When only one dimension is specified, the number of bins (rows) is L_1/b_1.

This array can be output with *fix ave/time*,

```
compute p all stress/cartesian x 0.1
fix 2 all ave/time 100 1 100 c_p[*] file dump_p.out mode vector
```

The values calculated by this compute are "intensive". The stress values will be in pressure *units*. The number density values are in inverse volume *units*.

NOTE 1: The local stress does not include any Lennard-Jones tail corrections to the stress added by the *pair_modify tail yes* command, since those are contributions to the global system pressure.

NOTE 2: The local stress profiles generated by these computes are similar to those obtained by the *method-of-planes (MOP)*. A key difference is that compute *stress/mop/profile* considers particles crossing a set of planes, while *stress/cartesian* computes averages for a set of small volumes. Moreover, this compute computes the diagonal components of the stress tensor P_{xx}, P_{yy}, and P_{zz}, while *stress/mop/profile* computes the components P_{ix}, P_{iy}, and P_{iz}, where i is the direction normal to the plane.

More information on the similarities and differences can be found in *(Ikeshoji)*.

3.136.5 Restrictions

These computes calculate the stress tensor contributions for pair and bond forces only (no angle, dihedral, improper, or kspace force). It requires pairwise force calculations not available for most many-body pair styles.

These computes are part of the EXTRA-COMPUTE package. They are only enabled if LAMMPS was built with that package. See the *Build package* doc page for more info.

3.136.6 Related commands

compute stress/atom, *compute pressure*, *compute stress/mop/profile*, *compute stress/spherical*, *compute stress/cylinder*

(Ikeshoji) Ikeshoji, Hafskjold, Furuholt, Mol Sim, 29, 101-109, (2003).

3.137 compute stress/cylinder command

3.138 compute stress/spherical command

3.138.1 Syntax

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

- ID, group-ID are documented in *compute* command
- style = stress/spherical or stress/cylinder
- args = argument specific to the compute style

stress/cylinder args = zlo zh Rmax bin_width keyword
  zlo = minimum z-boundary for cylinder
  zhi = maximum z-boundary for cylinder
  Rmax = maximum radius to perform calculation to
  bin_width = width of radial bins to use for calculation
  keyword = ke (zero or one can be specified)
    ke = yes or no
stress/spherical
  x0, y0, z0 = origin of the spherical coordinate system
  bin_width = width of spherical shells
  Rmax = maximum radius of spherical shells

3.138.2 Examples

```
compute 1 all stress/cylinder -10.0 10.0 15.0 0.25
compute 1 all stress/cylinder -10.0 10.0 15.0 0.25 ke no
compute 1 all stress/spherical 0 0 0 0.1 10
```

3.138.3 Description

Compute *stress/cylinder*, and compute *stress/spherical* define computations that calculate profiles of the diagonal components of the local stress tensor in the specified coordinate system. The stress tensor is split into a kinetic contribution P^k and a virial contribution P^v. The sum gives the total stress tensor $P = P^k + P^v$. These computes can for example be used to calculate the diagonal components of the local stress tensor of surfaces with cylindrical or spherical symmetry. These computes obeys momentum balance through fluid interfaces. They use the Irving–Kirkwood contour, which is the straight line between particle pairs.

The compute *stress/cylinder* computes the stress profile along the radial direction in cylindrical coordinates, as described in *(Addington)*. The compute *stress/spherical* computes the stress profile along the radial direction in spherical coordinates, as described in *(Ikeshoji)*.

3.138.4 Output info

The default output columns for *stress/cylinder* are the radius to the center of the cylindrical shell, number density, P^k_{rr}, $P^k_{\phi\phi}$, P^k_{zz}, P^v_{rr}, $P^v_{\phi\phi}$, and P^v_{zz}. When the keyword *ke* is set to *no*, the kinetic contributions are not calculated, and consequently there are only 5 columns: the position of the center of the cylindrical shell, the number density, P^v_{rr}, $P^v_{\phi\phi}$, and P^v_{zz}. The number of bins (rows) is $R_{\max}/b$, where b is the specified bin width.

The output columns for *stress/spherical* are the position of the center of the spherical shell, the number density, P^k_{rr}, $P^k_{\theta\theta}$, $P^k_{\phi\phi}$, P^v_{rr}, $P^v_{\theta\theta}$, and $P^v_{\phi\phi}$. There are 8 columns and the number of bins (rows) is $R_{\max}/b$, where b is the specified bin width.

This array can be output with *fix ave/time*,

```
compute 1 all stress/spherical 0 0 0 0.1 10
fix 2 all ave/time 100 1 100 c_p[*] file dump_p.out mode vector
```

The values calculated by this compute are "intensive". The stress values will be in pressure *units*. The number density values are in inverse volume *units*.

NOTE 1: The local stress does not include any Lennard-Jones tail corrections to the stress added by the *pair_modify tail yes* command, since those are contributions to the global system pressure.

3.138.5 Restrictions

These computes calculate the stress tensor contributions for pair styles only (i.e., no bond, angle, dihedral, etc. contributions, and in the presence of bonded interactions, the result may be incorrect due to exclusions for *special bonds* excluding pairs of atoms completely). It requires pairwise force calculations not available for most many-body pair styles. Note that *k*-space calculations are also excluded.

These computes are part of the EXTRA-COMPUTE package. They are only enabled if LAMMPS was built with that package. See the *Build package* doc page for more info.

3.138.6 Related commands

compute stress/atom, *compute pressure*, *compute stress/mop/profile*, *compute stress/cartesian*

3.138.7 Default

The keyword default for ke in style *stress/cylinder* is yes.

(Ikeshoji) Ikeshoji, Hafskjold, Furuholt, Mol Sim, 29, 101-109, (2003).

(Addington) Addington, Long, Gubbins, J Chem Phys, 149, 084109 (2018).

3.139 compute stress/mop command

3.140 compute stress/mop/profile command

3.140.1 Syntax

```
compute ID group-ID style dir args keywords ...
```

- ID, group-ID are documented in *compute* command
- style = *stress/mop* or *stress/mop/profile*
- dir = *x* or *y* or *z* is the direction normal to the plane
- args = argument specific to the compute style
- keywords = *kin* or *conf* or *total* or *pair* or *bond* or *angle* or *dihedral* (one or more can be specified)

stress/mop args = pos
  pos = lower or center or upper or coordinate value (distance units) is the position of the plane
stress/mop/profile args = origin delta
  origin = lower or center or upper or coordinate value (distance units) is the position of the first plane
  delta = value (distance units) is the distance between planes

3.140.2 Examples

```
compute 1 all stress/mop x lower total
compute 1 liquid stress/mop z 0.0 kin conf
fix 1 all ave/time 10 1000 10000 c_1[*] file mop.time
fix 1 all ave/time 10 1000 10000 c_1[2] file mop.time

compute 1 all stress/mop/profile x lower 0.1 total
compute 1 liquid stress/mop/profile z 0.0 0.25 kin conf
fix 1 all ave/time 500 20 10000 c_1[*] ave running overwrite file mopp.time mode vector
```

3.140.3 Description

Compute *stress/mop* and compute *stress/mop/profile* define computations that calculate components of the local stress tensor using the method of planes *(Todd)*. Specifically in compute *stress/mop* calculates 3 components are computed in directions *dir*,*x*; *dir*,*y*; and *dir*,*z*; where *dir* is the direction normal to the plane, while in compute *stress/mop/profile* the profile of the stress is computed.

Contrary to methods based on histograms of atomic stress (i.e., using *compute stress/atom*), the method of planes is compatible with mechanical balance in heterogeneous systems and at interfaces *(Todd)*.

The stress tensor is the sum of a kinetic term and a configurational term, which are given respectively by Eq. (21) and Eq. (16) in *(Todd)*. For the kinetic part, the algorithm considers that atoms have crossed the plane if their positions at times $t - \Delta t$ and t are one on either side of the plane, and uses the velocity at time $t - \Delta t/2$ given by the velocity Verlet algorithm.

Added in version 15Jun2023: contributions from bond, angle and dihedral potentials

Between one and seven keywords can be used to indicate which contributions to the stress must be computed: total stress (total), kinetic stress (kin), configurational stress (conf), stress due to bond stretching (bond), stress due to angle bending (angle), stress due to dihedral terms (dihedral) and/or due to pairwise non-bonded interactions (pair).

NOTE 1: The configurational stress is computed considering all pairs of atoms where at least one atom belongs to group group-ID.

NOTE 2: The local stress does not include any Lennard-Jones tail corrections to the stress added by the *pair_modify tail yes* command, since those are contributions to the global system pressure.

NOTE 3: The local stress profile generated by compute *stress/mop/profile* is similar to that obtained by compute *stress/cartesian*. A key difference is that compute *stress/mop/profile* considers particles crossing a set of planes, while *stress/cartesian* computes averages for a set of small volumes. Moreover, *stress/cartesian* compute computes the diagonal components of the stress tensor P_{xx}, P_{yy}, and P_{zz}, while *stress/mop/profile* computes the components P_{ix}, P_{iy}, and P_{iz}, where i is the direction normal to the plane.

3.140.4 Output info

Compute *stress/mop* calculates a global vector (indices starting at 1), with 3 values for each declared keyword (in the order the keywords have been declared). For each keyword, the stress tensor components are ordered as follows: stress_dir,x, stress_dir,y, and stress_dir,z.

Compute *stress/mop/profile* instead calculates a global array, with 1 column giving the position of the planes where the stress tensor was computed, and with 3 columns of values for each declared keyword (in the order the keywords have been declared). For each keyword, the profiles of stress tensor components are ordered as follows: stress_dir,x; stress_dir,y; and stress_dir,z.

The values are in pressure *units*.

The values produced by this compute can be accessed by various *output commands*. For instance, the results can be written to a file using the *fix ave/time* command. Please see the example in the examples/PACKAGES/mop folder.

3.140.5 Restrictions

These styles are part of the EXTRA-COMPUTE package. They are only enabled if LAMMPS is built with that package. See the *Build package* doc page on for more info.

The method is implemented for orthogonal simulation boxes whose size does not change in time, and axis-aligned planes.

The method only works with two-body pair interactions, because it requires the class method Pair::single() to be implemented, which is not possible for manybody potentials. In particular, compute *stress/mop/profile* and *stress/mop* do not work with more than two-body pair interactions, long range (kspace) interactions and improper intramolecular interactions.

The impact of fixes that affect the stress (e.g. fix langevin) is also not included in the stress computed here.

3.140.6 Related commands

compute stress/atom, *compute pressure*, *compute stress/cartesian*, *compute stress/cylinder*, *compute stress/spherical*

3.140.7 Default

none

(Todd) B. D. Todd, Denis J. Evans, and Peter J. Daivis: "Pressure tensor for inhomogeneous fluids", Phys. Rev. E 52, 1627 (1995).

(Ikeshoji) Ikeshoji, Hafskjold, Furuholt, Mol Sim, 29, 101-109, (2003).

3.141 compute force/tally command

3.142 compute heat/flux/tally command

3.143 compute heat/flux/virial/tally command

3.144 compute pe/tally command

3.145 compute pe/mol/tally command

3.146 compute stress/tally command

3.146.1 Syntax

```
compute ID group-ID style group2-ID
```

- ID, group-ID are documented in *compute* command
- style = *force/tally* or *heat/flux/tally* or *heat/flux/virial/tally* or *pe/tally* or *pe/mol/tally* or *stress/tally*

- group2-ID = group ID of second (or same) group

3.146.2 Examples

```
compute 1 lower force/tally upper
compute 1 left pe/tally right
compute 1 lower stress/tally lower
compute 1 subregion heat/flux/tally all
compute 1 liquid heat/flux/virial/tally solid
```

3.146.3 Description

Define a computation that calculates properties between two groups of atoms by accumulating them from pairwise non-bonded computations. Except for *heat/flux/virial/tally*, the two groups can be the same. This is similar to *compute group/group* only that the data is accumulated directly during the non-bonded force computation. The computes *force/tally*, *pe/tally*, *stress/tally*, and *heat/flux/tally* are primarily provided as example how to program additional, more sophisticated computes using the tally callback mechanism. Compute *pe/mol/tally* is one such style, that can—through using this mechanism—separately tally intermolecular and intramolecular energies. Something that would otherwise be impossible without integrating this as a core functionality into the base classes of LAMMPS.

Compute *heat/flux/tally* obtains the heat flux (strictly speaking, heat flow) inside the first group, which is the sum of the convective contribution due to atoms in the first group and the virial contribution due to interaction between the first and second groups:

$$\mathbf{Q} = \sum_{i \in \text{group 1}} e_i \mathbf{v}_i + \frac{1}{2} \sum_{i \in \text{group 1}} \sum_{\substack{j \in \text{group 2} \\ j \neq i}} \left(\mathbf{F}_{ij} \cdot \mathbf{v}_j \right) \mathbf{r}_{ij}$$

When the second group in *heat/flux/tally* is set to "all", the resulting values will be identical to that obtained by *compute heat/flux*, provided only pairwise interactions exist.

Compute *heat/flux/virial/tally* obtains the total virial heat flux (strictly speaking, heat flow) into the first group due to interaction with the second group, and is defined as:

$$Q = \frac{1}{2} \sum_{i \in \text{group 1}} \sum_{j \in \text{group 2}} \mathbf{F}_{ij} \cdot \left(\mathbf{v}_i + \mathbf{v}_j \right)$$

Although, the *heat/flux/virial/tally* compute does not include the convective term, it can be used to obtain the total heat flux over control surfaces, when there are no particles crossing over, such as is often in solid–solid and solid–liquid interfaces. This would be identical to the method of planes method. Note that the *heat/flux/virial/tally* compute is distinctly different from the *heat/flux* and *heat/flux/tally* computes, that are essentially volume averaging methods. The following example demonstrates the difference:

```
# System with only pairwise interactions.
# Non-periodic boundaries in the x direction.
# Has LeftLiquid and RightWall groups along x direction.

# Heat flux over the solid-liquid interface
compute hflow_hfvt RightWall heat/flux/virial/tally LeftLiquid
variable hflux_hfvt equal c_hflow_hfvt/(ly*lz)
```

(continues on next page)

(continued from previous page)

```
# x component of approximate heat flux vector inside the liquid region,
# two approaches.
#
compute myKE all ke/atom
compute myPE all pe/atom
compute myStress all stress/atom NULL virial
compute hflow_hf LeftLiquid heat/flux myKE myPE myStress
variable hflux_hf equal c_hflow_hf[1]/${volLiq}
#
compute hflow_hft LeftLiquid heat/flux/tally all
variable hflux_hft equal c_hflow_hft[1]/${volLiq}

# Pressure over the solid-liquid interface, three approaches.
#
compute force_gg RightWall group/group LeftLiquid
variable press_gg equal c_force_gg[1]/(ly*lz)
#
compute force_ft RightWall force/tally LeftLiquid
compute rforce_ft RightWall reduce sum c_force_ft[1]
variable press_ft equal c_rforce_ft/(ly*lz)
#
compute rforce_hfvt all reduce sum c_hflow_hfvt[1]
variable press_hfvt equal c_rforce_hfvt/(ly*lz)
```

The pairwise contributions are computing via a callback that the compute registers with the non-bonded pairwise force computation. This limits the use to systems that have no bonds, no Kspace, and no many-body interactions. On the other hand, the computation does not have to compute forces or energies a second time and thus can be much more efficient. The callback mechanism allows to write more complex pairwise property computations.

3.146.4 Output info

- Compute *pe/tally* calculates a global scalar (the energy) and a per atom scalar (the contributions of the single atom to the global scalar).
- Compute *pe/mol/tally* calculates a global four-element vector containing (in this order): *evdwl* and *ecoul* for intramolecular pairs and *evdwl* and *ecoul* for intermolecular pairs. Since molecules are identified by their molecule IDs, the partitioning does not have to be related to molecules, but the energies are tallied into the respective slots depending on whether the molecule IDs of a pair are the same or different.
- Compute *force/tally* calculates a global scalar (the force magnitude) and a per atom 3-element vector (force contribution from each atom).
- Compute *stress/tally* calculates a global scalar (average of the diagonal elements of the stress tensor) and a per atom vector (the six elements of stress tensor contributions from the individual atom).
- As in *compute heat/flux*, compute *heat/flux/tally* calculates a global vector of length 6, where the first three components are the x, y, z components of the full heat flow vector, and the next three components are the corresponding components of just the convective portion of the flow (i.e., the first term in the equation for $\mathbf{Q}$).
- Compute *heat/flux/virial/tally* calculates a global scalar (heat flow) and a per atom three-element vector (contribution to the force acting over atoms in the first group from individual atoms in both groups).

Both the scalar and vector values calculated by this compute are "extensive".

3.146.5 Restrictions

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

Not all pair styles can be evaluated in a pairwise mode as required by this compute. For example, 3-body and other many-body potentials, such as *Tersoff* and *Stillinger-Weber* cannot be used. *EAM* potentials only include the pair potential portion of the EAM interaction when used by this compute, not the embedding term. Also bonded or Kspace interactions do not contribute to this compute.

These computes are not compatible with accelerated pair styles from the GPU, INTEL, KOKKOS, or OPENMP packages. They will either create an error or print a warning when required data was not tallied in the required way and thus the data acquisition functions from these computes not called.

When used with dynamic groups, a *run 0* command needs to be inserted in order to initialize the dynamic groups before accessing the computes.

3.146.6 Related commands

- *compute group/group*
- *compute heat/flux*

3.146.7 Default

none

3.147 compute tdpd/cc/atom command

3.147.1 Syntax

```
compute ID group-ID tdpd/cc/atom index
```

- ID, group-ID are documented in *compute* command
- tdpd/cc/atom = style name of this compute command
- index = index of chemical species (1 to Nspecies)

3.147.2 Examples

```
compute 1 all tdpd/cc/atom 2
```

3.147.3 Description

Define a computation that calculates the per-atom chemical concentration of a specified species for each tDPD particle in a group.

The chemical concentration of each species is defined as the number of molecules carried by a tDPD particle for dilute solution. For more details see *(Li2015)*.

3.147.4 Output info

This compute calculates a per-atom vector, which can be accessed by any command that uses per-atom values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

The per-atom vector values will be in the units of chemical species per unit mass.

3.147.5 Restrictions

This compute is part of the DPD-MESO package. It is only enabled if LAMMPS was built with that package. See the *Build package* page for more info.

3.147.6 Related commands

pair_style tdpd

3.147.7 Default

none

(Li2015) Li, Yazdani, Tartakovsky, Karniadakis, J Chem Phys, 143: 014101 (2015). DOI: 10.1063/1.4923254

3.148 compute temp command

Accelerator Variants: *temp/kk*

3.148.1 Syntax

```
compute ID group-ID temp
```

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

3.148.2 Examples

```
compute 1 all temp
compute myTemp mobile temp
```

3.148.3 Description

Define a computation that calculates the temperature of a group of atoms. A compute of this style can be used by any command that computes a temperature, e.g. *thermo_modify*, *fix temp/rescale*, *fix npt*, etc.

The temperature is calculated by the formula

$$T = \frac{2E_{\rm kin}}{N_{\rm DOF}k_B} \quad \text{with} \quad E_{\rm kin} = \sum_{i=1}^{N_{\rm atoms}} \frac{1}{2} m_i v_i^2 \quad \text{and} \quad N_{\rm DOF} = n_{\rm dim} N_{\rm atoms} - n_{\rm dim} - N_{\rm fixDOFs}$$

where $E_{\rm kin}$ is the total kinetic energy of the group of atoms, $n_{\rm dim}$ is the dimensionality of the simulation (i.e. either 2 or 3), $N_{\rm atoms}$ is the number of atoms in the group, $N_{\rm fixDOFs}$ is the number of degrees of freedom removed by fix commands (see below), k_B is the Boltzmann constant, and T is the resulting computed temperature.

A symmetric tensor, stored as a six-element vector, is also calculated by this compute for use in the computation of a pressure tensor by the *compute pressue* command. The formula for the components of the tensor is the same as the above expression for $E_{\rm kin}$, except that the 1/2 factor is NOT included and the v_i^2 is replaced by $v_{i,x}v_{i,y}$ for the xy component, and so on. Note that because it lacks the 1/2 factor, these tensor components are twice those of the traditional kinetic energy tensor. The six components of the vector are ordered xx, yy, zz, xy, xz, yz.

The number of atoms contributing to the temperature is assumed to be constant for the duration of the run; use the *dynamic* option of the *compute_modify* command if this is not the case.

This compute subtracts out degrees-of-freedom due to fixes that constrain molecular motion, such as *fix shake* and *fix rigid*. This means the temperature of groups of atoms that include these constraints will be computed correctly. If needed, the subtracted degrees-of-freedom can be altered using the *extra* option of the *compute_modify* command. By default this *extra* component is initialized to $n_{\rm dim}$ (as shown in the formula above) to represent the degrees of freedom removed from a system due to its translation invariance due to periodic boundary conditions.

A compute of this style with the ID of "thermo_temp" is created when LAMMPS starts up, as if this command were in the input script:

```
compute thermo_temp all temp
```

See the "thermo_style" command for more details.

See the *Howto thermostat* page for a discussion of different ways to compute temperature and perform thermostatting.

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.

3.148.4 Output info

This compute calculates a global scalar (the temperature) and a global vector of length six (symmetric tensor), which can be accessed by indices 1–6. These values can be used by any command that uses global scalar or vector values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

The scalar value calculated by this compute is "intensive". The vector values are "extensive".

The scalar value is in temperature *units*. The vector values are in energy *units*.

3.148.5 Restrictions

none

3.148.6 Related commands

compute temp/partial, *compute temp/region*, *compute pressure*

3.148.7 Default

none

3.149 compute temp/asphere command

3.149.1 Syntax

```
compute ID group-ID temp/asphere keyword value ...
```

- ID, group-ID are documented in *compute* command
- temp/asphere = style name of this compute command
- zero or more keyword/value pairs may be appended
- keyword = *bias* or *dof*

```
bias value = bias-ID
  bias-ID = ID of a temperature compute that removes a velocity bias
dof value = all or rotate
  all = compute temperature of translational and rotational degrees of freedom
  rotate = compute temperature of just rotational degrees of freedom
```

3.149.2 Examples

```
compute 1 all temp/asphere
compute myTemp mobile temp/asphere bias tempCOM
compute myTemp mobile temp/asphere dof rotate
```

3.149.3 Description

Define a computation that calculates the temperature of a group of aspherical particles, including a contribution from both their translational and rotational kinetic energy. This differs from the usual *compute temp* command, which assumes point particles with only translational kinetic energy.

Only finite-size particles (aspherical or spherical) can be included in the group. For 3d finite-size particles, each has six degrees of freedom (three translational, three rotational). For 2d finite-size particles, each has three degrees of freedom (two translational, one rotational).

Note

This choice for degrees of freedom (DOF) assumes that all finite-size aspherical or spherical particles in your model will freely rotate, sampling all their rotational DOF. It is possible to use a combination of interaction potentials and fixes that induce no torque or otherwise constrain some of all of your particles so that this is not the case. Then there are fewer DOF and you should use the *compute_modify extra/dof* command to adjust the DOF accordingly.

For example, an aspherical particle with all three of its shape parameters the same is a sphere. If it does not rotate, then it should have 3 DOF instead of 6 in 3d (or two instead of three in 2d). A uniaxial aspherical particle has two of its three shape parameters the same. If it does not rotate around the axis perpendicular to its circular cross section, then it should have 5 DOF instead of 6 in 3d. The latter is the case for uniaxial ellipsoids in a *GayBerne model* since there is no induced torque around the optical axis. It will also be the case for biaxial ellipsoids when exactly two of the semiaxes have the same length and the corresponding relative well depths are equal.

The translational kinetic energy is computed the same as is described by the *compute temp* command. The rotational kinetic energy is computed as $\frac{1}{2} I \omega^2$, where I is the inertia tensor for the aspherical particle and ω is its angular velocity, which is computed from its angular momentum.

Note

For *2d models*, particles are treated as ellipsoids, not ellipses, meaning their moments of inertia will be the same as in 3d.

A kinetic energy tensor, stored as a six-element vector, is also calculated by this compute. The formula for the components of the tensor is the same as the above formula, except that v^2 and ω^2 are replaced by $v_x v_y$ and $\omega_x \omega_y$ for the xy component, and the appropriate elements of the moment of inertia tensor are used. The six components of the vector are ordered xx, yy, zz, xy, xz, yz.

A symmetric tensor, stored as a six-element vector, is also calculated by this compute for use in the computation of a pressure tensor by the *compute pressue* command. The formula for the components of the tensor is the same as the above expression for E_{kin}, except that the 1/2 factor is NOT included and the v_i^2 and ω^2 are replaced by $v_x v_y$ and $\omega_x \omega_y$ for the xy component, and so on. And the appropriate elements of the moment of inertia tensor are used. Note that because it lacks the 1/2 factor, these tensor components are twice those of the traditional kinetic energy tensor. The six components of the vector are ordered xx, yy, zz, xy, xz, yz.

The number of atoms contributing to the temperature is assumed to be constant for the duration of the run; use the *dynamic/dof* option of the *compute_modify* command if this is not the case.

This compute subtracts out translational degrees-of-freedom due to fixes that constrain molecular motion, such as *fix shake* and *fix rigid*. This means the temperature of groups of atoms that include these constraints will be computed correctly. If needed, the subtracted degrees-of-freedom can be altered using the *extra/dof* option of the *compute_modify* command.

See the *Howto thermostat* page for a discussion of different ways to compute temperature and perform thermostatting.

The keyword/value option pairs are used in the following ways.

For the *bias* keyword, *bias-ID* refers to the ID of a temperature compute that removes a "bias" velocity from each atom. This allows compute temp/sphere to compute its thermal temperature after the translational kinetic energy components have been altered in a prescribed way (e.g., to remove a flow velocity profile). Thermostats that use this compute will work with this bias term. See the doc pages for individual computes that calculate a temperature and the doc pages for fixes that perform thermostatting for more details.

For the *dof* keyword, a setting of *all* calculates a temperature that includes both translational and rotational degrees of freedom. A setting of *rotate* calculates a temperature that includes only rotational degrees of freedom.

3.149.4 Output info

This compute calculates a global scalar (the temperature) and a global vector of length 6 (symmetric tensor), which can be accessed by indices 1–6. These values can be used by any command that uses global scalar or vector values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

The scalar value calculated by this compute is "intensive". The vector values are "extensive".

The scalar value is in temperature *units*. The vector values are in energy *units*.

3.149.5 Restrictions

This compute 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 compute requires that atoms store 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.

3.149.6 Related commands

compute temp

3.149.7 Default

none

3.150 compute temp/body command

3.150.1 Syntax

```
compute ID group-ID temp/body keyword value ...
```

- ID, group-ID are documented in *compute* command
- temp/body = style name of this compute command
- zero or more keyword/value pairs may be appended
- keyword = *bias* or *dof*

  bias value = bias-ID
    bias-ID = ID of a temperature compute that removes a velocity bias
  dof value = all or rotate
    all = compute temperature of translational and rotational degrees of freedom
    rotate = compute temperature of just rotational degrees of freedom

3.150.2 Examples

```
compute 1 all temp/body
compute myTemp mobile temp/body bias tempCOM
compute myTemp mobile temp/body dof rotate
```

3.150.3 Description

Define a computation that calculates the temperature of a group of body particles, including a contribution from both their translational and rotational kinetic energy. This differs from the usual *compute temp* command, which assumes point particles with only translational kinetic energy.

Only body particles can be included in the group. For 3d particles, each has 6 degrees of freedom (3 translational, 3 rotational). For 2d body particles, each has 3 degrees of freedom (2 translational, 1 rotational).

Note

This choice for degrees of freedom (DOF) assumes that all body particles in your model will freely rotate, sampling all their rotational DOF. It is possible to use a combination of interaction potentials and fixes that induce no torque or otherwise constrain some of all of your particles so that this is not the case. Then there are less DOF and you should use the *compute_modify extra/dof* command to adjust the DOF accordingly.

The translational kinetic energy is computed the same as is described by the *compute temp* command. The rotational kinetic energy is computed as $\frac{1}{2} I \omega^2$, where I is the moment of inertia tensor for the aspherical particle and ω is its angular velocity, which is computed from its angular momentum.

A symmetric tensor, stored as a six-element vector, is also calculated by this compute for use in the computation of a pressure tensor by the *compute pressue* command. The formula for the components of the tensor is the same as the above expression for $E_{\rm kin}$, except that the 1/2 factor is NOT included and the v_i^2 and ω^2 are replaced by $v_x v_y$ and $\omega_x \omega_y$ for the *xy* component, and so on. And the appropriate elements of the moment of inertia tensor are used. Note that because it lacks the 1/2 factor, these tensor components are twice those of the traditional kinetic energy tensor. The six components of the vector are ordered *xx*, *yy*, *zz*, *xy*, *xz*, *yz*.

The number of atoms contributing to the temperature is assumed to be constant for the duration of the run; use the *dynamic/dof* option of the *compute_modify* command if this is not the case.

This compute subtracts out translational degrees-of-freedom due to fixes that constrain molecular motion, such as *fix shake* and *fix rigid*. This means the temperature of groups of atoms that include these constraints will be computed correctly. If needed, the subtracted degrees-of-freedom can be altered using the *extra/dof* option of the *compute_modify* command.

See the *Howto thermostat* page for a discussion of different ways to compute temperature and perform thermostatting.

The keyword/value option pairs are used in the following ways.

For the *bias* keyword, *bias-ID* refers to the ID of a temperature compute that removes a "bias" velocity from each atom. This allows compute temp/sphere to compute its thermal temperature after the translational kinetic energy components have been altered in a prescribed way (e.g., to remove a flow velocity profile). Thermostats that use this compute will work with this bias term. See the doc pages for individual computes that calculate a temperature and the doc pages for fixes that perform thermostatting for more details.

For the *dof* keyword, a setting of *all* calculates a temperature that includes both translational and rotational degrees of freedom. A setting of *rotate* calculates a temperature that includes only rotational degrees of freedom.

3.150.4 Output info

This compute calculates a global scalar (the temperature) and a global vector of length 6 (symmetric tensor), which can be accessed by indices 1–6. These values can be used by any command that uses global scalar or vector values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

The scalar value calculated by this compute is "intensive". The vector values are "extensive".

The scalar value is in temperature *units*. The vector values are in energy *units*.

3.150.5 Restrictions

This compute 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 compute requires that atoms store angular momentum and a quaternion as defined by the *atom_style body* command.

3.150.6 Related commands

compute temp

3.150.7 Default

none

3.151 compute temp/chunk command

3.151.1 Syntax

```
compute ID group-ID temp/chunk chunkID value1 value2 ... keyword value ...
```

- ID, group-ID are documented in *compute* command
- temp/chunk = style name of this compute command
- chunkID = ID of *compute chunk/atom* command
- zero or more values can be listed as value1,value2,etc.
- value = *temp* or *kecom* or *internal*

```
temp = temperature of each chunk
kecom = kinetic energy of each chunk based on velocity of center of mass
internal = internal kinetic energy of each chunk
```

- zero or more keyword/value pairs may be appended
- keyword = *com* or *bias* or *adof* or *cdof*

```
com value = yes or no
  yes = subtract center-of-mass velocity from each chunk before calculating temperature
  no = do not subtract center-of-mass velocity
bias value = bias-ID
  bias-ID = ID of a temperature compute that removes a velocity bias
adof value = dof_per_atom
  dof_per_atom = define this many degrees-of-freedom per atom
cdof value = dof_per_chunk
  dof_per_chunk = define this many degrees-of-freedom per chunk
```

3.151.2 Examples

```
compute 1 fluid temp/chunk molchunk
compute 1 fluid temp/chunk molchunk temp internal
compute 1 fluid temp/chunk molchunk bias tpartial adof 2.0
```

3.151.3 Description

Define a computation that calculates the temperature of a group of atoms that are also in chunks, after optionally subtracting out the center-of-mass velocity of each chunk. By specifying optional values, it can also calculate the per-chunk temperature or energies of the multiple chunks of atoms.

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* and *Howto chunk* doc pages for details of how chunks can be defined and examples of how they can be used to measure properties of a system.

The temperature is calculated by the formula

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

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

The DOF is calculated as $N \times adof + N_{\mathrm{chunk}} \times cdof$, where N is the number of atoms contributing to the kinetic energy, *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.

A symmetric tensor, stored as a six-element vector, is also calculated by this compute. The formula for the components of the tensor is the same as the above expression for E_{kin}, except that the 1/2 factor is NOT included and the v_i^2 is replaced by $v_{i,x}v_{i,y}$ for the *xy* component, and so on. Note that because it lacks the 1/2 factor, these tensor components are twice those of the traditional kinetic energy tensor. The six components of the vector are ordered *xx*, *yy*, *zz*, *xy*, *xz*, *yz*.

Note that the number of atoms contributing to the temperature is calculated each time the temperature is evaluated since it is assumed the atoms may be dynamically assigned to chunks. Thus there is no need to use the *dynamic* option of the *compute_modify* command for this compute style.

If any optional values are specified, then per-chunk quantities are also calculated and stored in a global array, as described below.

The *temp* value calculates the temperature for each chunk by the formula

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

where KE is the total kinetic energy of the chunk of atoms (sum of $\frac{1}{2}mv^2$), DOF is the 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 number of degrees of freedom (DOF) in this case is calculated as $N \times adof + cdof$, 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, *cdof* = 2 or 3 = dimensionality of system, as set via the *dimension* command, and *cdof* = 0.0. This gives the usual formula for temperature.

The *kecom* value calculates the kinetic energy of each chunk as if all its atoms were moving with the velocity of the center-of-mass of the chunk.

The *internal* value calculates the internal kinetic energy of each chunk. The interal KE is summed over the atoms in the chunk using an internal "thermal" velocity for each atom, which is its velocity minus the center-of-mass velocity of the chunk.

Note that currently the global and per-chunk temperatures calculated by this compute only include 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 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.

Note that the per-chunk temperature calculated by this compute and the *fix ave/chunk temp* command can be different. This compute calculates the temperature for each chunk for a single snapshot. Fix ave/chunk 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 timesteps were collected together as one set of atoms to calculate their temperature. This compute allows the center-of-mass velocity of each chunk to be subtracted before calculating the temperature; fix ave/chunk does not.

Note

Only atoms in the specified group contribute to the calculations performed by this compute. The *compute chunk/atom* command defines its own group; atoms will have a chunk ID = 0 if they are not in that group, signifying they are not assigned to a chunk, and will thus also not contribute to this calculation. You can specify the "all" group for this command if you simply want to include atoms with non-zero chunk IDs.

The simplest way to output the per-chunk results of the compute temp/chunk calculation to a file is to use the *fix ave/time* command, for example:

```
compute cc1 all chunk/atom molecule
compute myChunk all temp/chunk cc1 temp
fix 1 all ave/time 100 1 100 c_myChunk[1] file tmp.out mode vector
```

The keyword/value option pairs are used in the following ways.

The *com* keyword can be used with a value of *yes* to subtract the velocity of the center-of-mass for each chunk from the velocity of the atoms in that chunk, before calculating either the global or per-chunk temperature. This can be useful if the atoms are streaming or otherwise moving collectively, and you wish to calculate only the thermal temperature.

For the *bias* keyword, *bias-ID* refers to the ID of a temperature compute that removes a "bias" velocity from each atom. This also allows calculation of the global or per-chunk temperature using only the thermal temperature of atoms in each chunk after the translational kinetic energy components have been altered in a prescribed way (e.g., to remove a velocity profile). It also applies to the calculation of the other per-chunk values, such as *kecom* or *internal*, which involve the center-of-mass velocity of each chunk, which is calculated after the velocity bias is removed from each atom. Note that the temperature compute will apply its bias globally to the entire system, not on a per-chunk basis.

The *adof* and *cdof* keywords define the values used in the degree of freedom (DOF) formulas used for the global or per-chunk temperature, as described above. 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 *cdof* could be set to the remaining degrees of freedom for the entire molecule (entire chunk in this case; i.e., 6 for 3d, or 3 for 2d, for a rigid molecule).

3.151.4 Output info

This compute calculates a global scalar (the temperature) and a global vector of length 6 (symmetric tensor), which can be accessed by indices 1–6. These values can be used by any command that uses global scalar or vector values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

This compute also optionally calculates a global array, if one or more of the optional values are specified. The number of rows in the array is the number of chunks *Nchunk* as calculated by the specified *compute chunk/atom* command. The number of columns is the number of specified values (1 or more). These values can be accessed by any command that uses global array values from a compute as input. Again, see the *Howto output* doc page for an overview of LAMMPS output options.

The scalar value calculated by this compute is "intensive". The vector values are "extensive". The array values are "intensive".

The scalar value is in temperature *units*. The vector values are in energy *units*. The array values will be in temperature *units* for the *temp* value, and in energy *units* for the *kecom* and *internal* values.

3.151.5 Restrictions

The *com* and *bias* keywords cannot be used together.

3.151.6 Related commands

compute temp, *fix ave/chunk temp*

3.151.7 Default

The option defaults are com no, no bias, adof = dimensionality of the system (2 or 3), and cdof = 0.0.

3.152 compute temp/com command

3.152.1 Syntax

```
compute ID group-ID temp/com
```

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

3.152.2 Examples

```
compute 1 all temp/com
compute myTemp mobile temp/com
```

3.152.3 Description

Define a computation that calculates the temperature of a group of atoms, after subtracting out the center-of-mass velocity of the group. This is useful if the group is expected to have a non-zero net velocity for some reason. A compute of this style can be used by any command that computes a temperature, (e.g., *thermo_modify*, *fix temp/rescale*, *fix npt*).

After the center-of-mass velocity has been subtracted from each atom, the temperature is calculated by the formula

$$\mathrm{KE} = \frac{\mathrm{dim}}{2} N k_B T,$$

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

A symmetric tensor, stored as a six-element vector, is also calculated by this compute for use in the computation of a pressure tensor by the *compute pressue* command. The formula for the components of the tensor is the same as the above expression for E_{kin}, except that the 1/2 factor is NOT included and the v_i^2 is replaced by $v_{i,x}v_{i,y}$ for the xy component, and so on. Note that because it lacks the 1/2 factor, these tensor components are twice those of the traditional kinetic energy tensor. The six components of the vector are ordered xx, yy, zz, xy, xz, yz.

The number of atoms contributing to the temperature is assumed to be constant for the duration of the run; use the *dynamic* option of the *compute_modify* command if this is not the case.

The removal of the center-of-mass velocity by this fix is essentially computing the temperature after a "bias" has been removed from the velocity of the atoms. If this compute is used with a fix command that performs thermostatting then this bias will be subtracted from each atom, thermostatting of the remaining thermal velocity will be performed, and the bias will be added back in. Thermostatting fixes that work in this way include *fix nvt*, *fix temp/rescale*, *fix temp/berendsen*, and *fix langevin*.

This compute subtracts out degrees-of-freedom due to fixes that constrain molecular motion, such as *fix shake* and *fix rigid*. This means the temperature of groups of atoms that include these constraints will be computed correctly. If needed, the subtracted degrees-of-freedom can be altered using the *extra* option of the *compute_modify* command.

See the *Howto thermostat* page for a discussion of different ways to compute temperature and perform thermostatting.

3.152.4 Output info

This compute calculates a global scalar (the temperature) and a global vector of length 6 (symmetric tensor), which can be accessed by indices 1–6. These values can be used by any command that uses global scalar or vector values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

The scalar value calculated by this compute is "intensive". The vector values are "extensive".

The scalar value is in temperature *units*. The vector values is in energy *units*.

3.152.5 Restrictions

none

3.152.6 Related commands

compute temp

3.152.7 Default

none

3.153 compute temp/cs command

3.153.1 Syntax

```
compute ID group-ID temp/cs group1 group2
```

- ID, group-ID are documented in *compute* command
- temp/cs = style name of this compute command
- group1 = group-ID of either cores or shells
- group2 = group-ID of either shells or cores

3.153.2 Examples

```
compute oxygen_c-s all temp/cs O_core O_shell
compute core_shells all temp/cs cores shells
```

3.153.3 Description

Define a computation that calculates the temperature of a system based on the center-of-mass velocity of atom pairs that are bonded to each other. This compute is designed to be used with the adiabatic core/shell model of *(Mitchell and Fincham)*. See the *Howto coreshell* page for an overview of the model as implemented in LAMMPS. Specifically, this compute enables correct temperature calculation and thermostatting of core/shell pairs where it is desirable for the internal degrees of freedom of the core/shell pairs to not be influenced by a thermostat. A compute of this style can be used by any command that computes a temperature via *fix_modify* (e.g., *fix temp/rescale*, *fix npt*).

Note that this compute does not require all ions to be polarized, hence defined as core/shell pairs. One can mix core/shell pairs and ions without a satellite particle if desired. The compute will consider the non-polarized ions according to the physical system.

For this compute, core and shell particles are specified by two respective group IDs, which can be defined using the *group* command. The number of atoms in the two groups must be the same and there should be one bond defined between a pair of atoms in the two groups. Non-polarized ions which might also be included in the treated system should not be included into either of these groups, they are taken into account by the *group-ID* (second argument) of the compute.

The temperature is calculated by the formula

$$\mathrm{KE} = \frac{\dim}{2} N k_B T,$$

where KE is the total kinetic energy of the group of atoms (sum of $\frac{1}{2}mv^2$), dim = 2 or 3 is the dimensionality of the simulation, N is the number of atoms in the group, k_B is the Boltzmann constant, and T is the absolute temperature. Note that the velocity of each core or shell atom used in the KE calculation is the velocity of the center-of-mass (COM) of the core/shell pair the atom is part of.

A symmetric tensor, stored as a six-element vector, is also calculated by this compute for use in the computation of a pressure tensor by the *compute pressue* command. The formula for the components of the tensor is the same as the above expression for $E_{\rm kin}$, except that the 1/2 factor is NOT included and the v_i^2 is replaced by $v_{i,x}v_{i,y}$ for the xy component, and so on. Note that because it lacks the 1/2 factor, these tensor components are twice those of the traditional kinetic energy tensor. The six components of the vector are ordered xx, yy, zz, xy, xz, yz.

The change this fix makes to core/shell atom velocities is essentially computing the temperature after a "bias" has been removed from the velocity of the atoms. This "bias" is the velocity of the atom relative to the center-of-mass velocity of the core/shell pair. If this compute is used with a fix command that performs thermostatting then this bias will be subtracted from each atom, thermostatting of the remaining center-of-mass velocity will be performed, and the bias will be added back in. This means the thermostatting will effectively be performed on the core/shell pairs, instead of on the individual core and shell atoms. Thermostatting fixes that work in this way include *fix nvt*, *fix temp/rescale*, *fix temp/berendsen*, and *fix langevin*.

The internal energy of core/shell pairs can be calculated by the *compute temp/chunk* command, if chunks are defined as core/shell pairs. See the *Howto coreshell* doc page for more discussion on how to do this.

3.153.4 Output info

This compute calculates a global scalar (the temperature) and a global vector of length 6 (symmetric tensor), which can be accessed by indices 1–6. These values can be used by any command that uses global scalar or vector values from a compute as input.

The scalar value calculated by this compute is "intensive". The vector values are "extensive".

The scalar value is in temperature *units*. The vector values are in energy *units*.

3.153.5 Restrictions

The number of core/shell pairs contributing to the temperature is assumed to be constant for the duration of the run. No fixes should be used which generate new molecules or atoms during a simulation.

3.153.6 Related commands

compute temp, *compute temp/chunk*

3.153.7 Default

none

(Mitchell and Fincham) Mitchell, Fincham, J Phys Condensed Matter, 5, 1031-1038 (1993).

3.154 compute temp/deform command

Accelerator Variants: *temp/deform/kk*

3.154.1 Syntax

```
compute ID group-ID temp/deform
```

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

3.154.2 Examples

```
compute myTemp all temp/deform
```

3.154.3 Description

Define a computation that calculates the temperature of a group of atoms, after subtracting out a streaming velocity induced by the simulation box changing size and/or shape, for example in a non-equilibrium MD (NEMD) simulation. The size/shape change is induced by use of the *fix deform* command. A compute of this style is created by the *fix nvt/sllod* command to compute the thermal temperature of atoms for thermostatting purposes. A compute of this style can also be used by any command that computes a temperature (e.g., *thermo_modify*, *fix temp/rescale*, *fix npt*).

The deformation fix changes the box size and/or shape over time, so each atom in the simulation box can be thought of as having a "streaming" velocity. For example, if the box is being sheared in x, relative to y, then atoms at the bottom of the box (low y) have a small x velocity, while atoms at the top of the box (high y) have a large x velocity. This position-dependent streaming velocity is subtracted from each atom's actual velocity to yield a thermal velocity, which is then used to compute the temperature.

Note

Fix deform has an option for remapping either atom coordinates or velocities to the changing simulation box. When using this compute in conjunction with a deforming box, fix deform should NOT remap atom positions, but rather should let atoms respond to the changing box by adjusting their own velocities (or let *fix deform* remap the atom velocities; see its remap option). If fix deform does remap atom positions, then they appear to move with the box but their velocity is not changed, and thus they do NOT have the streaming velocity assumed by this compute. LAMMPS will warn you if fix deform is defined and its remap setting is not consistent with this compute.

After the streaming velocity has been subtracted from each atom, the temperature is calculated by the formula

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

where KE is the total kinetic energy of the group of atoms (sum of $\frac{1}{2}mv^2$, dim = 2 or 3 is the dimensionality of the simulation, N is the number of atoms in the group, k_B is the Boltzmann constant, and T is the temperature. Note that v in the kinetic energy formula is the atom's velocity.

A symmetric tensor, stored as a six-element vector, is also calculated by this compute for use in the computation of a pressure tensor by the *compute pressue* command. The formula for the components of the tensor is the same as the above expression for E_{kin}, except that the 1/2 factor is NOT included and the v_i^2 is replaced by $v_{i,x}v_{i,y}$ for the xy component,

and so on. Note that because it lacks the 1/2 factor, these tensor components are twice those of the traditional kinetic energy tensor. The six components of the vector are ordered *xx*, *yy*, *zz*, *xy*, *xz*, *yz*.

The number of atoms contributing to the temperature is assumed to be constant for the duration of the run; use the *dynamic* option of the *compute_modify* command if this is not the case.

The removal of the box deformation velocity component by this fix is essentially computing the temperature after a "bias" has been removed from the velocity of the atoms. If this compute is used with a fix command that performs thermostatting then this bias will be subtracted from each atom, thermostatting of the remaining thermal velocity will be performed, and the bias will be added back in. Thermostatting fixes that work in this way include *fix nvt*, *fix temp/rescale*, *fix temp/berendsen*, and *fix langevin*.

Note

The temperature calculated by this compute is only accurate if the atoms are indeed moving with a stream velocity profile that matches the box deformation. If not, then the compute will subtract off an incorrect stream velocity, yielding a bogus thermal temperature. You should **not** assume that your atoms are streaming at the same rate the box is deforming. Rather, you should monitor their velocity profiles (e.g., via the *fix ave/chunk* command). You can also compare the results of this compute to *compute temp/profile*, which actually calculates the stream profile before subtracting it. If the two computes do not give roughly the same temperature, then your atoms are not streaming consistent with the box deformation. See the *fix deform* command for more details on ways to get atoms to stream consistently with the box deformation.

This compute subtracts out degrees-of-freedom due to fixes that constrain molecular motion, such as *fix shake* and *fix rigid*. This means the temperature of groups of atoms that include these constraints will be computed correctly. If needed, the subtracted degrees-of-freedom can be altered using the *extra* option of the *compute_modify* command.

See the *Howto thermostat* page for a discussion of different ways to compute temperature and perform thermostatting.

3.154.4 Output info

This compute calculates a global scalar (the temperature) and a global vector of length 6 (symmetric tensor), which can be accessed by indices 1–6. These values can be used by any command that uses global scalar or vector values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

The scalar value calculated by this compute is "intensive". The vector values are "extensive".

The scalar value is in temperature *units*. The vector values are in energy *units*.

3.154.5 Restrictions

none

3.154.6 Related commands

compute temp/ramp, *compute temp/profile*, *fix deform*, *fix nvt/sllod*

3.154.7 Default

none

3.155 compute temp/deform/eff command

3.155.1 Syntax

```
compute ID group-ID temp/deform/eff
```

- ID, group-ID are documented in *compute* command
- temp/deform/eff = style name of this compute command

3.155.2 Examples

```
compute myTemp all temp/deform/eff
```

3.155.3 Description

Define a computation that calculates the temperature of a group of nuclei and electrons in the *electron force field* model, after subtracting out a streaming velocity induced by the simulation box changing size and/or shape, for example in a non-equilibrium MD (NEMD) simulation. The size/shape change is induced by use of the *fix deform* command. A compute of this style is created by the *fix nvt/sllod/eff* command to compute the thermal temperature of atoms for thermostatting purposes. A compute of this style can also be used by any command that computes a temperature (e.g., *thermo_modify*, *fix npt/eff*).

The calculation performed by this compute is exactly like that described by the *compute temp/deform* command, except that the formulas for the temperature (scalar) and diagonal components of the symmetric tensor (vector) include the radial electron velocity contributions, as discussed by the *compute temp/eff* command. Note that only the translational degrees of freedom for each nuclei or electron are affected by the streaming velocity adjustment. The radial velocity component of the electrons is not affected.

3.155.4 Output info

This compute calculates a global scalar (the temperature) and a global vector of length 6 (symmetric tensor), which can be accessed by indices 1–6. These values can be used by any command that uses global scalar or vector values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

The scalar value calculated by this compute is “intensive”. The vector values are “extensive”.

The scalar value is in temperature *units*. The vector values are in energy *units*.

3.155.5 Restrictions

This compute 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.

3.155.6 Related commands

compute temp/ramp, *fix deform*, *fix nvt/sllod/eff*

3.155.7 Default

none

3.156 compute temp/drude command

3.156.1 Syntax

```
compute ID group-ID temp/drude
```

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

3.156.2 Examples

```
compute TDRUDE all temp/drude
```

Example input scripts available: examples/PACKAGES/drude.

3.156.3 Description

Define a computation that calculates the temperatures of core–Drude pairs. This compute 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 consist of a core particle and a Drude particle connected by a harmonic bond, and the relative motion of these Drude oscillators is usually maintained cold by a specific thermostat that acts on the relative motion of the core–Drude particle pairs. Therefore, because LAMMPS considers Drude particles as normal atoms in its default temperature compute (*compute temp* command), the reduced temperature of the core–Drude particle pairs is not calculated correctly.

By contrast, this compute calculates the temperature of the cores using center-of-mass velocities of the core–Drude pairs, and the reduced temperature of the Drude particles using the relative velocities of the Drude particles with respect to their cores. Non-polarizable atoms are considered as cores. Their velocities contribute to the temperature of the cores.

3.156.4 Output info

This compute calculates a global scalar (the temperature) and a global vector of length 6, which can be accessed by indices 1–6, whose components are

1. temperature of the centers of mass (temperature units)
2. temperature of the dipoles (temperature units)
3. number of degrees of freedom of the centers of mass
4. number of degrees of freedom of the dipoles
5. kinetic energy of the centers of mass (energy units)
6. kinetic energy of the dipoles (energy units)

These values can be used by any command that uses global scalar or vector values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

Both the scalar value and the first two values of the vector calculated by this compute are "intensive". The other four vector values are "extensive".

3.156.5 Restrictions

The number of degrees of freedom contributing to the temperature is assumed to be constant for the duration of the run unless the *fix_modify command* sets the option *dynamic/dof yes*.

3.156.6 Related commands

fix drude, *fix langevin/drude*, *fix drude/transform*, *pair_style thole*, *compute temp*

3.156.7 Default

none

3.157 compute temp/eff command

3.157.1 Syntax

```
compute ID group-ID temp/eff
```

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

3.157.2 Examples

```
compute 1 all temp/eff
compute myTemp mobile temp/eff
```

3.157.3 Description

Define a computation that calculates the temperature of a group of nuclei and electrons in the *electron force field* model. A compute of this style can be used by commands that compute a temperature (e.g., *thermo_modify*, *fix npt/eff*).

The temperature is calculated by the formula

$$\mathrm{KE} = \frac{\mathrm{dim}}{2} N k_B T,$$

where KE is the total kinetic energy of the group of atoms (sum of $\frac{1}{2}mv^2$ for nuclei and sum of $\frac{1}{2}(mv^2 + \frac{3}{4}ms^2)$ for electrons, where s includes the radial electron velocity contributions), dim = 2 or 3 is the dimensionality of the simulation, N is the number of atoms (only total number of nuclei in the eFF (see the *pair_eff* command) in the group, k_B is the Boltzmann constant, and T is the absolute temperature. This expression is summed over all nuclear and electronic degrees of freedom, essentially by setting the kinetic contribution to the heat capacity to $\frac{3}{2}k$ (where only nuclei contribute). This subtlety is valid for temperatures well below the Fermi temperature, which for densities two to five times the density of liquid hydrogen ranges from 86,000 to 170,000 K.

Note

For eFF models, in order to override the default temperature reported by LAMMPS in the thermodynamic quantities reported via the *thermo* command, the user should apply a *thermo_modify* command, as shown in the following example:

```
compute         effTemp all temp/eff
thermo_style    custom step etotal pe ke temp press
thermo_modify   temp effTemp
```

A six-component kinetic energy tensor is also calculated by this compute for use in the computation of a pressure tensor. The formula for the components of the tensor is the same as the above formula, except that v^2 is replaced by $v_x v_y$ for the xy component, etc. For the eFF, again, the radial electronic velocities are also considered.

The number of atoms contributing to the temperature is assumed to be constant for the duration of the run; use the *dynamic* option of the *compute_modify* command if this is not the case.

This compute subtracts out degrees-of-freedom due to fixes that constrain molecular motion, such as *fix shake* and *fix rigid*. This means the temperature of groups of atoms that include these constraints will be computed correctly. If needed, the subtracted degrees-of-freedom can be altered using the *extra* option of the *compute_modify* command.

See the *Howto thermostat* page for a discussion of different ways to compute temperature and perform thermostatting.

3.157.4 Output info

The scalar value calculated by this compute is "intensive", meaning it is independent of the number of atoms in the simulation. The vector values are "extensive", meaning they scale with the number of atoms in the simulation.

3.157.5 Restrictions

This compute 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.

3.157.6 Related commands

compute temp/partial, *compute temp/region*, *compute pressure*

3.157.7 Default

none

3.158 compute temp/partial command

3.158.1 Syntax

```
compute ID group-ID temp/partial xflag yflag zflag
```

- ID, group-ID are documented in *compute* command
- temp/partial = style name of this compute command
- xflag,yflag,zflag = 0/1 for whether to exclude/include this dimension

3.158.2 Examples

```
compute newT flow temp/partial 1 1 0
```

3.158.3 Description

Define a computation that calculates the temperature of a group of atoms, after excluding one or more velocity components. A compute of this style can be used by any command that computes a temperature (e.g. *thermo_modify*, *fix temp/rescale*, *fix npt*).

The temperature is calculated by the formula

$$\mathrm{KE} = \frac{\mathrm{dim}}{2} N k_B T,$$

where KE is the total kinetic energy of the group of atoms (sum of $\frac{1}{2}mv^2$), dim = 2 or 3 is the dimensionality of the simulation, N is the number of atoms in the group, k_B is the Boltzmann constant, and T = temperature. The calculation of KE excludes the x, y, or z dimensions if *xflag*, *yflag*, or *zflag* is 0. The dim parameter is adjusted to give the correct number of degrees of freedom.

A symmetric tensor, stored as a six-element vector, is also calculated by this compute for use in the computation of a pressure tensor by the *compute pressue* command. The formula for the components of the tensor is the same as the above expression for $E_{\rm kin}$, except that the 1/2 factor is NOT included and the v_i^2 is replaced by $v_{i,x}v_{i,y}$ for the *xy* component, and so on. Note that because it lacks the 1/2 factor, these tensor components are twice those of the traditional kinetic energy tensor. The six components of the vector are ordered *xx*, *yy*, *zz*, *xy*, *xz*, *yz*.

The number of atoms contributing to the temperature is assumed to be constant for the duration of the run; use the *dynamic* option of the *compute_modify* command if this is not the case.

The removal of velocity components by this fix is essentially computing the temperature after a "bias" has been removed from the velocity of the atoms. If this compute is used with a fix command that performs thermostatting then this bias will be subtracted from each atom, thermostatting of the remaining thermal velocity will be performed, and the bias will be added back in. Thermostatting fixes that work in this way include *fix nvt*, *fix temp/rescale*, *fix temp/berendsen*, and *fix langevin*.

This compute subtracts out degrees-of-freedom due to fixes that constrain molecular motion, such as *fix shake* and *fix rigid*. This means the temperature of groups of atoms that include these constraints will be computed correctly. If needed, the subtracted degrees-of-freedom can be altered using the *extra* option of the *compute_modify* command.

See the *Howto thermostat* page for a discussion of different ways to compute temperature and perform thermostatting.

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.

3.158.4 Output info

This compute calculates a global scalar (the temperature) and a global vector of length 6 (symmetric tensor), which can be accessed by indices 1–6. These values can be used by any command that uses global scalar or vector values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

The scalar value calculated by this compute is "intensive". The vector values are "extensive".

The scalar value is in temperature *units*. The vector values are in energy *units*.

3.158.5 Restrictions

none

3.158.6 Related commands

compute temp, *compute temp/region*, *compute pressure*

3.158.7 Default

none

3.159 compute temp/profile command

3.159.1 Syntax

```
compute ID group-ID temp/profile xflag yflag zflag binstyle args
```

- ID, group-ID are documented in *compute* command
- temp/profile = style name of this compute command
- xflag,yflag,zflag = 0/1 for whether to exclude/include this dimension
- binstyle = *x* or *y* or *z* or *xy* or *yz* or *xz* or *xyz*

  x arg = Nx
  y arg = Ny
  z arg = Nz
  xy args = Nx Ny
  yz args = Ny Nz
  xz args = Nx Nz
  xyz args = Nx Ny Nz
    Nx, Ny, Nz = number of velocity bins in x, y, z dimensions
- zero or more keyword/value pairs may be appended
- keyword = *out*

  out value = tensor or bin

3.159.2 Examples

```
compute myTemp flow temp/profile 1 1 1 x 10
compute myTemp flow temp/profile 1 1 1 x 10 out bin
compute myTemp flow temp/profile 0 1 1 xyz 20 20 20
```

3.159.3 Description

Define a computation that calculates the temperature of a group of atoms, after subtracting out a spatially-averaged center-of-mass velocity field, before computing the kinetic energy. This can be useful for thermostatting a collection of atoms undergoing a complex flow (e.g. via a profile-unbiased thermostat (PUT) as described in *(Evans)*). A compute of this style can be used by any command that computes a temperature (e.g. *thermo_modify*, *fix temp/rescale*, *fix npt*).

The *xflag*, *yflag*, *zflag* settings determine which components of average velocity are subtracted out.

The *binstyle* setting and its *Nx*, *Ny*, *Nz* arguments determine how bins are setup to perform spatial averaging. "Bins" can be 1d slabs, 2d pencils, or 3d bricks depending on which *binstyle* is used. The simulation box is partitioned conceptually into $Nx \times Ny \times Nz$ bins. Depending on the *binstyle*, you may only specify one or two of these values; the others are effectively set to 1 (no binning in that dimension). For non-orthogonal (triclinic) simulation boxes, the bins are "tilted" slabs or pencils or bricks that are parallel to the tilted faces of the box. See the *region prism* command for a discussion of the geometry of tilted boxes in LAMMPS.

When a temperature is computed, the center-of-mass velocity for the set of atoms that are both in the compute group and in the same spatial bin is calculated. This bias velocity is then subtracted from the velocities of individual atoms in the bin to yield a thermal velocity for each atom. Note that if there is only one atom in the bin, its thermal velocity will thus be 0.0.

After the spatially-averaged velocity field has been subtracted from each atom, the temperature is calculated by the formula

$$\mathrm{KE} = \left(\frac{\mathrm{dim}}{N} - N_s N_x N_y N_z - \mathrm{extra} \right) \frac{k_B T}{2},$$

where KE is the total kinetic energy of the group of atoms (sum of $\frac{1}{2} m v^2$; dim = 2 or 3 is the dimensionality of the simulation; N_s = 0, 1, 2, or 3 for streaming velocity subtracted in 0, 1, 2, or 3 dimensions, respectively; *extra* is the number of extra degrees of freedom; N is the number of atoms in the group; k_B is the Boltzmann constant, and T is the absolute temperature. The $N_s N_x N_y N_z$ term is the number of degrees of freedom subtracted to adjust for the removal of the center-of-mass velocity in each direction of the *Nx*Ny*Nz* bins, as discussed in the *(Evans)* paper. The extra term defaults to dim $- N_s$ and accounts for overall conservation of center-of-mass velocity across the group in directions where streaming velocity is *not* subtracted. This can be altered using the *extra* option of the *compute_modify* command.

If the *out* keyword is used with a *tensor* value, which is the default, then a symmetric tensor, stored as a six-element vector, is also calculated by this compute for use in the computation of a pressure tensor by the *compute pressue* command. The formula for the components of the tensor is the same as the above expression for E_{kin}, except that the 1/2 factor is NOT included and the v_i^2 is replaced by $v_{i,x} v_{i,y}$ for the *xy* component, and so on. Note that because it lacks the 1/2 factor, these tensor components are twice those of the traditional kinetic energy tensor. The six components of the vector are ordered *xx*, *yy*, *zz*, *xy*, *xz*, *yz*.

If the *out* keyword is used with a *bin* value, the count of atoms and computed temperature for each bin are stored for output, as an array of values, as described below. The temperature of each bin is calculated as described above, where the bias velocity is subtracted and only the remaining thermal velocity of atoms in the bin contributes to the temperature. See the note below for how the temperature is normalized by the degrees-of-freedom of atoms in the bin.

The number of atoms contributing to the temperature is assumed to be constant for the duration of the run; use the *dynamic* option of the *compute_modify* command if this is not the case.

The removal of the spatially-averaged velocity field by this fix is essentially computing the temperature after a "bias" has been removed from the velocity of the atoms. If this compute is used with a fix command that performs thermostatting then this bias will be subtracted from each atom, thermostatting of the remaining thermal velocity will be performed, and the bias will be added back in. Thermostatting fixes that work in this way include *fix nvt*, *fix temp/rescale*, *fix temp/berendsen*, and *fix langevin*.

This compute subtracts out degrees-of-freedom due to fixes that constrain molecular motion, such as *fix shake* and *fix rigid*. This means the temperature of groups of atoms that include these constraints will be computed correctly. If needed, the subtracted degrees-of-freedom can be altered using the *extra* option of the *compute_modify* command.

Note

When using the *out* keyword with a value of *bin*, the calculated temperature for each bin includes the degrees-of-freedom adjustment described in the preceding paragraph for fixes that constrain molecular motion, as well as the adjustment due to the *extra* option (which defaults to *dim* - *Ns* as described above), by fractionally applying them based on the fraction of atoms in each bin. As a result, the bin degrees-of-freedom summed over all bins exactly equals the degrees-of-freedom used in the scalar temperature calculation, $\Sigma N_{\mathrm{DOF}_i} = N_{\mathrm{DOF}}$ and the corresponding relation for temperature is also satisfied ($\Sigma N_{\mathrm{DOF}_i} T_i = N_{\mathrm{DOF}} T$). These relations will break down in cases for which the adjustment exceeds the actual number of degrees of freedom in a bin. This could happen if a bin is empty or in situations in which rigid molecules are non-uniformly distributed, in which case the reported temperature within a bin may not be accurate.

See the *Howto thermostat* page for a discussion of different ways to compute temperature and perform thermostatting. Using this compute in conjunction with a thermostatting fix, as explained there, will effectively implement a profile-unbiased thermostat (PUT), as described in *(Evans)*.

3.159.4 Output info

This compute calculates a global scalar (the temperature). Depending on the setting of the *out* keyword, it also calculates a global vector or array. For *out* = *tensor*, it calculates a vector of length 6 (symmetric tensor), which can be accessed by indices 1–6. For *out* = *bin* it calculates a global array which has 2 columns and N rows, where N is the number of bins. The first column contains the number of atoms in that bin. The second contains the temperature of that bin, calculated as described above. The ordering of rows in the array is as follows. Bins in x vary fastest, then y, then z. Thus for a $10 \times 10 \times 10$ 3d array of bins, there will be 1000 rows. The bin with indices $(i_x, i_y, i_z) = (2, 3, 4)$ would map to row $M = 10^2(i_z - 1) + 10(i_y - 1) + i_x = 322$, where the rows are numbered from 1 to 1000 and the bin indices are numbered from 1 to 10 in each dimension.

These values can be used by any command that uses global scalar or vector or array values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

The scalar value calculated by this compute is “intensive”. The vector values are “extensive”. The array values are “intensive”.

The scalar value us in temperature *units*. The vector values are in energy *units*. The first column of array values are counts; the values in the second column will be in temperature *units*.

3.159.5 Restrictions

You should not use too large a velocity-binning grid, especially in 3d. In the current implementation, the binned velocity averages are summed across all processors, so this will be inefficient if the grid is too large, and the operation is performed every timestep, as it will be for most thermostats.

3.159.6 Related commands

compute temp, *compute temp/ramp*, *compute temp/deform*, *compute pressure*

3.159.7 Default

The option default is out = tensor.

(Evans) Evans and Morriss, Phys Rev Lett, 56, 2172-2175 (1986).

3.160 compute temp/ramp command

3.160.1 Syntax

```
compute ID group-ID temp/ramp vdim vlo vhi dim clo chi keyword value ...
```

- ID, group-ID are documented in *compute* command
- temp/ramp = style name of this compute command
- vdim = *vx* or *vy* or *vz*
- vlo,vhi = subtract velocities between vlo and vhi (velocity units)
- dim = *x* or *y* or *z*
- clo,chi = lower and upper bound of domain to subtract from (distance units)
- zero or more keyword/value pairs may be appended
- keyword = *units*

units value = lattice or box

3.160.2 Examples

```
compute 2nd middle temp/ramp vx 0 8 y 2 12 units lattice
```

3.160.3 Description

Define a computation that calculates the temperature of a group of atoms, after subtracting out an ramped velocity profile before computing the kinetic energy. A compute of this style can be used by any command that computes a temperature (e.g. *thermo_modify*, *fix temp/rescale*, *fix npt*).

The meaning of the arguments for this command which define the velocity ramp are the same as for the *velocity ramp* command which was presumably used to impose the velocity.

After the ramp velocity has been subtracted from the specified dimension for each atom, the temperature is calculated by the formula

$$\mathrm{KE} = \frac{\mathrm{dim}}{2} N k_B T,$$

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

The *units* keyword determines the meaning of the distance units used for coordinates (*clo*, *chi*) and velocities (*vlo*, *vhi*). 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 (i.e., velocity in lattice spacings per unit time). The *lattice* command must have been previously used to define the lattice spacing.

A symmetric tensor, stored as a six-element vector, is also calculated by this compute for use in the computation of a pressure tensor by the *compute pressue* command. The formula for the components of the tensor is the same as the above expression for $E_{\rm kin}$, except that the 1/2 factor is NOT included and the v_i^2 is replaced by $v_{i,x}v_{i,y}$ for the *xy* component, and so on. Note that because it lacks the 1/2 factor, these tensor components are twice those of the traditional kinetic energy tensor. The six components of the vector are ordered *xx*, *yy*, *zz*, *xy*, *xz*, *yz*.

The number of atoms contributing to the temperature is assumed to be constant for the duration of the run; use the *dynamic* option of the *compute_modify* command if this is not the case.

The removal of the ramped velocity component by this fix is essentially computing the temperature after a "bias" has been removed from the velocity of the atoms. If this compute is used with a fix command that performs thermostatting then this bias will be subtracted from each atom, thermostatting of the remaining thermal velocity will be performed, and the bias will be added back in. Thermostatting fixes that work in this way include *fix nvt*, *fix temp/rescale*, *fix temp/berendsen*, and *fix langevin*.

This compute subtracts out degrees-of-freedom due to fixes that constrain molecular motion, such as *fix shake* and *fix rigid*. This means the temperature of groups of atoms that include these constraints will be computed correctly. If needed, the subtracted degrees-of-freedom can be altered using the *extra* option of the *compute_modify* command.

See the *Howto thermostat* page for a discussion of different ways to compute temperature and perform thermostatting.

3.160.4 Output info

This compute calculates a global scalar (the temperature) and a global vector of length 6 (symmetric tensor), which can be accessed by indices 1–6. These values can be used by any command that uses global scalar or vector values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

The scalar value calculated by this compute is "intensive". The vector values are "extensive".

The scalar value is in temperature *units*. The vector values are in energy *units*.

3.160.5 Restrictions

none

3.160.6 Related commands

compute temp, *compute temp/profile*, *compute temp/deform*, *compute pressure*

3.160.7 Default

The option default is units = lattice.

3.161 compute temp/region command

3.161.1 Syntax

```
compute ID group-ID temp/region region-ID
```

- ID, group-ID are documented in *compute* command
- temp/region = style name of this compute command
- region-ID = ID of region to use for choosing atoms

3.161.2 Examples

```
compute mine flow temp/region boundary
```

3.161.3 Description

Define a computation that calculates the temperature of a group of atoms in a geometric region. This can be useful for thermostatting one portion of the simulation box. For example, a McDLT simulation where one side is cooled, and the other side is heated. A compute of this style can be used by any command that computes a temperature (e.g., *thermo_modify*, *fix temp/rescale*).

Note that a *region*-style temperature can be used to thermostat with *fix temp/rescale* or *fix langevin*, but should probably not be used with Nose–Hoover style fixes (*fix nvt*, *fix npt*, or *fix nph*) if the degrees of freedom included in the computed temperature vary with time.

The temperature is calculated by the formula

$$\mathrm{KE} = \frac{\mathrm{dim}}{2} N k_B T,$$

where KE = is the total kinetic energy of the group of atoms (sum of $\frac{1}{2}mv^2$), dim = 2 or 3 is the dimensionality of the simulation, N is the number of atoms in both the group and region, k_B is the Boltzmann constant, and T temperature.

A symmetric tensor, stored as a six-element vector, is also calculated by this compute for use in the computation of a pressure tensor by the *compute pressue* command. The formula for the components of the tensor is the same as the above expression for E_{kin}, except that the 1/2 factor is NOT included and the v_i^2 is replaced by $v_{i,x}v_{i,y}$ for the xy component, and so on. Note that because it lacks the 1/2 factor, these tensor components are twice those of the traditional kinetic energy tensor. The six components of the vector are ordered xx, yy, zz, xy, xz, yz.

The number of atoms contributing to the temperature is calculated each time the temperature is evaluated since it is assumed atoms can enter/leave the region. Thus there is no need to use the *dynamic* option of the *compute_modify* command for this compute style.

The removal of atoms outside the region by this fix is essentially computing the temperature after a "bias" has been removed, which in this case is the velocity of any atoms outside the region. If this compute is used with a fix command that performs thermostatting then this bias will be subtracted from each atom, thermostatting of the remaining thermal velocity will be performed, and the bias will be added back in. Thermostatting fixes that work in this way include *fix nvt*, *fix temp/rescale*, *fix temp/berendsen*, and *fix langevin*. This means that when this compute is used to calculate the temperature for any of the thermostatting fixes via the *fix modify temp* command, the thermostat will operate only on atoms that are currently in the geometric region.

Unlike other compute styles that calculate temperature, this compute does not subtract out degrees-of-freedom due to fixes that constrain motion, such as *fix shake* and *fix rigid*. This is because those degrees of freedom (e.g., a constrained

bond) could apply to sets of atoms that straddle the region boundary, and hence the concept is somewhat ill-defined. If needed the number of subtracted degrees of freedom can be set explicitly using the *extra* option of the *compute_modify* command.

See the *Howto thermostat* page for a discussion of different ways to compute temperature and perform thermostatting.

3.161.4 Output info

This compute calculates a global scalar (the temperature) and a global vector of length 6 (symmetric tensor), which can be accessed by indices 1–6. These values can be used by any command that uses global scalar or vector values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

The scalar value calculated by this compute is "intensive". The vector values are "extensive".

The scalar value is in temperature *units*. The vector values are in energy *units*.

3.161.5 Restrictions

none

3.161.6 Related commands

compute temp, *compute pressure*

3.161.7 Default

none

3.162 compute temp/region/eff command

3.162.1 Syntax

```
compute ID group-ID temp/region/eff region-ID
```

- ID, group-ID are documented in *compute* command
- temp/region/eff = style name of this compute command
- region-ID = ID of region to use for choosing atoms

3.162.2 Examples

```
compute mine flow temp/region/eff boundary
```

3.162.3 Description

Define a computation that calculates the temperature of a group of nuclei and electrons in the *electron force field* model, within a geometric region using the electron force field. A compute of this style can be used by commands that compute a temperature (e.g., *thermo_modify*).

The operation of this compute is exactly like that described by the *compute temp/region* command, except that the formulas for the temperature (scalar) and diagonal components of the symmetric tensor (vector) include the radial electron velocity contributions, as discussed by the *compute temp/eff* command.

3.162.4 Output info

This compute calculates a global scalar (the temperature) and a global vector of length 6 (symmetric tensor), which can be accessed by indices 1–6. These values can be used by any command that uses global scalar or vector values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

The scalar value calculated by this compute is "intensive". The vector values are "extensive".

The scalar value is in temperature *units*. The vector values are in energy *units*.

3.162.5 Restrictions

This compute 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.

3.162.6 Related commands

compute temp/region, *compute temp/eff* , *compute pressure*

3.162.7 Default

none

3.163 compute temp/rotate command

3.163.1 Syntax

```
compute ID group-ID temp/rotate
```

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

3.163.2 Examples

```
compute Tbead bead temp/rotate
```

3.163.3 Description

Define a computation that calculates the temperature of a group of atoms, after subtracting out the center-of-mass velocity and angular velocity of the group. This is useful if the group is expected to have a non-zero net velocity and/or global rotation motion for some reason. A compute of this style can be used by any command that computes a temperature (e.g., *thermo_modify*, *fix temp/rescale*, *fix npt*).

After the center-of-mass velocity and angular velocity has been subtracted from each atom, the temperature is calculated by the formula

$$\mathrm{KE} = \frac{\mathrm{dim}}{2} N k_B T,$$

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

A symmetric tensor, stored as a six-element vector, is also calculated by this compute for use in the computation of a pressure tensor by the *compute pressue* command. The formula for the components of the tensor is the same as the above expression for E_{kin}, except that the 1/2 factor is NOT included and the v_i^2 is replaced by $v_{i,x}v_{i,y}$ for the xy component, and so on. Note that because it lacks the 1/2 factor, these tensor components are twice those of the traditional kinetic energy tensor. The six components of the vector are ordered xx, yy, zz, xy, xz, yz.

The number of atoms contributing to the temperature is assumed to be constant for the duration of the run; use the *dynamic* option of the *compute_modify* command if this is not the case.

The removal of the center-of-mass velocity and angular velocity by this fix is essentially computing the temperature after a "bias" has been removed from the velocity of the atoms. If this compute is used with a fix command that performs thermostatting then this bias will be subtracted from each atom, thermostatting of the remaining thermal velocity will be performed, and the bias will be added back in. Thermostatting fixes that work in this way include *fix nvt*, *fix temp/rescale*, *fix temp/berendsen*, and *fix langevin*.

This compute subtracts out degrees-of-freedom due to fixes that constrain molecular motion, such as *fix shake* and *fix rigid*. This means the temperature of groups of atoms that include these constraints will be computed correctly. If needed, the subtracted degrees-of-freedom can be altered using the *extra* option of the *compute_modify* command.

See the *Howto thermostat* page for a discussion of different ways to compute temperature and perform thermostatting.

3.163.4 Output info

This compute calculates a global scalar (the temperature) and a global vector of length 6 (symmetric tensor), which can be accessed by indices 1-6. These values can be used by any command that uses global scalar or vector values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

The scalar value calculated by this compute is "intensive". The vector values are "extensive".

The scalar value is in temperature *units*. The vector values are in energy *units*.

3.163.5 Restrictions

This compute is part of the EXTRA-COMPUTE package. It is only enabled if LAMMPS was built with that package. See the *Build package* page for more info.

3.163.6 Related commands

compute temp

3.163.7 Default

none

3.164 compute temp/sphere command

3.164.1 Syntax

```
compute ID group-ID temp/sphere keyword value ...
```

- ID, group-ID are documented in *compute* command
- temp/sphere = style name of this compute command
- zero or more keyword/value pairs may be appended
- keyword = *bias* or *dof*

  bias value = bias-ID
    bias-ID = ID of a temperature compute that removes a velocity bias
  dof value = all or rotate
    all = compute temperature of translational and rotational degrees of freedom
    rotate = compute temperature of just rotational degrees of freedom

3.164.2 Examples

```
compute 1 all temp/sphere
compute myTemp mobile temp/sphere bias tempCOM
compute myTemp mobile temp/sphere dof rotate
```

3.164.3 Description

Define a computation that calculates the temperature of a group of spherical particles, including a contribution from both their translational and rotational kinetic energy. This differs from the usual *compute temp* command, which assumes point particles with only translational kinetic energy.

Both point and finite-size particles can be included in the group. Point particles do not rotate, so they have only three translational degrees of freedom. For 3d spherical particles, each has six degrees of freedom (three translational, three rotational). For 2d spherical particles, each has three degrees of freedom (two translational, one rotational).

Note

This choice for degrees of freedom (DOF) assumes that all finite-size spherical particles in your model will freely rotate, sampling all their rotational DOF. It is possible to use a combination of interaction potentials and fixes that induce no torque or otherwise constrain some of all of your particles so that this is not the case. Then there are less DOF and you should use the *compute_modify extra/dof* command to adjust the DOF accordingly.

The translational kinetic energy is computed the same as is described by the *compute temp* command. The rotational kinetic energy is computed as $\frac{1}{2} I \omega^2$, where I is the moment of inertia for a sphere and ω is the particle's angular velocity.

Note

For *2d models*, particles are treated as spheres, not disks, meaning their moment of inertia will be the same as in 3d.

A kinetic energy tensor, stored as a six-element vector, is also calculated by this compute. The formula for the components of the tensor is the same as the above formulas, except that v^2 and ω^2 are replaced by $v_x v_y$ and $\omega_x \omega_y$ for the xy component. The six components of the vector are ordered xx, yy, zz, xy, xz, yz.

A symmetric tensor, stored as a six-element vector, is also calculated by this compute for use in the computation of a pressure tensor by the *compute pressue* command. The formula for the components of the tensor is the same as the above expression for E_{kin}, except that the 1/2 factor is NOT included and the v_i^2 and ω^2 are replaced by $v_x v_y$ and $\omega_x \omega_y$ for the xy component, and so on. Note that because it lacks the 1/2 factor, these tensor components are twice those of the traditional kinetic energy tensor. The six components of the vector are ordered xx, yy, zz, xy, xz, yz.

The number of atoms contributing to the temperature is assumed to be constant for the duration of the run; use the *dynamic* option of the *compute_modify* command if this is not the case.

This compute subtracts out translational degrees-of-freedom due to fixes that constrain molecular motion, such as *fix shake* and *fix rigid*. This means the temperature of groups of atoms that include these constraints will be computed correctly. If needed, the subtracted degrees of freedom can be altered using the *extra/dof* option of the *compute_modify* command.

See the *Howto thermostat* page for a discussion of different ways to compute temperature and perform thermostatting.

The keyword/value option pairs are used in the following ways.

For the *bias* keyword, *bias-ID* refers to the ID of a temperature compute that removes a "bias" velocity from each atom. This allows compute temp/sphere to compute its thermal temperature after the translational kinetic energy components have been altered in a prescribed way (e.g., to remove a flow velocity profile). Thermostats that use this compute will work with this bias term. See the doc pages for individual computes that calculate a temperature and the doc pages for fixes that perform thermostatting for more details.

For the *dof* keyword, a setting of *all* calculates a temperature that includes both translational and rotational degrees of freedom. A setting of *rotate* calculates a temperature that includes only rotational degrees of freedom.

3.164.4 Output info

This compute calculates a global scalar (the temperature) and a global vector of length 6 (symmetric tensor), which can be accessed by indices 1–6. These values can be used by any command that uses global scalar or vector values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

The scalar value calculated by this compute is "intensive". The vector values are "extensive".

The scalar value is in temperature *units*. The vector values are in energy *units*.

3.164.5 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, or point particles with radius = 0.0.

3.164.6 Related commands

compute temp, *compute temp/asphere*

3.164.7 Default

The option defaults are no bias and dof = all.

3.165 compute temp/uef command

3.165.1 Syntax

```
compute ID group-ID temp/uef
```

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

3.165.2 Examples

```
compute 1 all temp/uef
compute 2 sel temp/uef
```

3.165.3 Description

This command is used to compute the kinetic energy tensor in the reference frame of the applied flow field when *fix nvt/uef* or *fix npt/uef* is used. It is not necessary to use this command to compute the scalar value of the temperature. A *compute temp* may be used for that purpose.

Output information for this command can be found in the documentation for *compute temp*.

3.165.4 Restrictions

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

This command can only be used when *fix nvt/uef* or *fix npt/uef* is active.

3.165.5 Related commands

compute temp, *fix nvt/uef* , *compute pressure/uef*

3.165.6 Default

none

3.166 compute ti command

3.166.1 Syntax

```
compute ID group ti keyword args ...
```

- ID, group-ID are documented in *compute* command
- ti = style name of this compute command
- one or more attribute/arg pairs may be appended
- keyword = pair style (lj/cut, gauss, born, etc.) or *tail* or *kspace*

  pair style args = atype v_name1 v_name2
    atype = atom type (see asterisk form below)
    v_name1 = variable with name1 that is energy scale factor and function of lambda
    v_name2 = variable with name2 that is derivative of v_name1 with respect to lambda
  tail args = atype v_name1 v_name2
    atype = atom type (see asterisk form below)
    v_name1 = variable with name1 that is energy tail correction scale factor and function of lambda
    v_name2 = variable with name2 that is derivative of v_name1 with respect to lambda
  kspace args = atype v_name1 v_name2
    atype = atom type (see asterisk form below)
    v_name1 = variable with name1 that is K-Space scale factor and function of lambda
    v_name2 = variable with name2 that is derivative of v_name1 with respect to lambda

3.166.2 Examples

```
compute 1 all ti lj/cut 1 v_lj v_dlj coul/long 2 v_c v_dc kspace 1 v_ks v_dks
compute 1 all ti lj/cut 1*3 v_lj v_dlj coul/long * v_c v_dc kspace * v_ks v_dks
```

3.166.3 Description

Define a computation that calculates the derivative of the interaction potential with respect to *lambda*, the coupling parameter used in a thermodynamic integration. This derivative can be used to infer a free energy difference resulting from an alchemical simulation, as described in *Eike*.

Typically this compute will be used in conjunction with the *fix adapt* command which can perform alchemical transformations by adjusting the strength of an interaction potential as a simulation runs, as defined by one or more *pair_style* or *kspace_style* commands. This scaling is done via a prefactor on the energy, forces, virial calculated by the pair or *k*-space style. The prefactor is often a function of a *lambda* parameter which may be adjusted from 0 to 1 (or vice versa) over the course of a *run*. The time-dependent adjustment is what the *fix adapt* command does.

Assume that the unscaled energy of a pair_style or kspace_style is given by U. Then the scaled energy is

$$U_s = f(\lambda)U$$

where f is some function of λ. What this compute calculates is

$$\frac{dU_s}{d\lambda} = U\frac{df(\lambda)}{d\lambda} = \frac{U_s}{f(\lambda)}\frac{df(\lambda)}{d\lambda},$$

which is the derivative of the system's scaled potential energy U_s with respect to λ.

To perform this calculation, you provide one or more atom types as *atype*. The variable *atype* can be specified in one of two ways. An explicit numeric value can be used, as in the first example above, or a wildcard asterisk can be used in place of or in conjunction with the *atype* argument to select multiple 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).

You also specify two functions, as *equal-style variables*. The first is specified as *v_name1*, where *name1* is the name of the variable, and is $f(\lambda)$ in the notation above. The second is specified as *v_name2*, where *name2* is the name of the variable, and is $df(\lambda)/d\lambda$ in the notation above (i.e., it is the analytic derivative of f with respect to λ). Note that the *name1* variable is also typically given as an argument to the *fix adapt* command.

An alchemical simulation may use several pair potentials together, invoked via the *pair_style hybrid or hybrid/overlay* command. The total $dU_s/d\lambda$ for the overall system is calculated as the sum of each contributing term as listed by the keywords in the *compute ti* command. Individual pair potentials can be listed, which will be sub-styles in the hybrid case. You can also include a *k*-space term via the *kspace* keyword. You can also include a pairwise long-range tail correction to the energy via the *tail* keyword.

For each term, you can specify a different (or the same) scale factor by the two variables that you list. Again, these will typically correspond toe the scale factors applied to these various potentials and the *k*-space contribution via the *fix adapt* command.

More details about the exact functional forms for the computation of du/dl can be found in the paper by *Eike*.

3.166.4 Output info

This compute calculates a global scalar, namely $dU_s/d\lambda$. This value can be used by any command that uses a global scalar value from a compute as input. See the *Howto output* doc page for an overview of LAMMPS output options.

The scalar value calculated by this compute is "extensive".

The scalar value will be in energy *units*.

3.166.5 Restrictions

This compute is part of the EXTRA-COMPUTE package. It is only enabled if LAMMPS was built with that package. See the *Build package* page for more info.

3.166.6 Related commands

fix adapt

3.166.7 Default

none

(Eike) Eike and Maginn, Journal of Chemical Physics, 124, 164503 (2006).

3.167 compute torque/chunk command

3.167.1 Syntax

```
compute ID group-ID torque/chunk chunkID
```

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

3.167.2 Examples

```
compute 1 fluid torque/chunk molchunk
```

3.167.3 Description

Define a computation that calculates the torque on multiple chunks of atoms.

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* and *Howto chunk* doc pages for details of how chunks can be defined and examples of how they can be used to measure properties of a system.

This compute calculates the three components of the torque vector for eqch chunk, due to the forces on the individual atoms in the chunk around the center-of-mass of the chunk. The calculation includes all effects due to atoms passing through periodic boundaries.

Note that only atoms in the specified group contribute to the calculation. The *compute chunk/atom* command defines its own group; atoms will have a chunk ID = 0 if they are not in that group, signifying they are not assigned to a chunk, and will thus also not contribute to this calculation. You can specify the "all" group for this command if you simply want to include atoms with non-zero chunk IDs.

Note

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

The simplest way to output the results of the compute torque/chunk calculation to a file is to use the *fix ave/time* command, for example:

```
compute cc1 all chunk/atom molecule
compute myChunk all torque/chunk cc1
fix 1 all ave/time 100 1 100 c_myChunk[*] file tmp.out mode vector
```

3.167.4 Output info

This compute calculates a global array where the number of rows is equal to the number of chunks *Nchunk* as calculated by the specified *compute chunk/atom* command. The number of columns is three for the x, y, and z components of the torque for each chunk. These values can be accessed by any command that uses global array values from a compute as input. See the *Howto output* doc page for an overview of LAMMPS output options.

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

3.167.5 Restrictions

none

3.167.6 Related commands

variable torque() function

3.167.7 Default

none

3.168 compute vacf command

3.168.1 Syntax

```
compute ID group-ID vacf
```

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

3.168.2 Examples

```
compute 1 all vacf
compute 1 upper vacf
```

3.168.3 Description

Define a computation that calculates the velocity auto-correlation function (VACF), averaged over a group of atoms. Each atom's contribution to the VACF is its current velocity vector dotted into its initial velocity vector at the time the compute was specified.

A vector of four quantities is calculated by this compute. The first three elements of the vector are $v_x v_{x,0}$ (and similar for the y and z components), summed and averaged over atoms in the group, where v_x is the current x-component of the velocity of the atom and $v_{x,0}$ is the initial x-component of the velocity of the atom. The fourth element of the vector is the total VACF (i.e., $(v_x v_{x,0} + v_y v_{y,0} + v_z v_{z,0})$), summed and averaged over atoms in the group.

The integral of the VACF versus time is proportional to the diffusion coefficient of the diffusing atoms. This can be computed in the following manner, using the *variable trap()* function:

```
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
```

Note

If you want the quantities calculated by this compute to be continuous when running from a *restart file*, then you should use the same ID for this compute, as in the original run. This is so that the fix this compute creates to store per-atom quantities will also have the same ID, and thus be initialized correctly with time=0 atom velocities from the restart file.

3.168.4 Output info

This compute calculates a global vector of length 4, which can be accessed by indices 1–4 by any command that uses global vector values from a compute as input. See the *Howto output* doc page for an overview of LAMMPS output options.

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

3.168.5 Restrictions

none

3.168.6 Related commands

compute msd

3.168.7 Default

none

3.169 compute vcm/chunk command

3.169.1 Syntax

```
compute ID group-ID vcm/chunk chunkID
```

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

3.169.2 Examples

```
compute 1 fluid vcm/chunk molchunk
```

3.169.3 Description

Define a computation that calculates the center-of-mass velocity for multiple chunks of atoms.

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* and *Howto chunk* doc pages for details of how chunks can be defined and examples of how they can be used to measure properties of a system.

This compute calculates the (x, y, z) components of the center-of-mass velocity for each chunk. This is done by summing mass*velocity for each atom in the chunk and dividing the sum by the total mass of the chunk.

Note that only atoms in the specified group contribute to the calculation. The *compute chunk/atom* command defines its own group; atoms will have a chunk ID = 0 if they are not in that group, signifying they are not assigned to a chunk,

and will thus also not contribute to this calculation. You can specify the "all" group for this command if you simply want to include atoms with non-zero chunk IDs.

The simplest way to output the results of the compute vcm/chunk calculation to a file is to use the *fix ave/time* command, for example:

```
compute cc1 all chunk/atom molecule
compute myChunk all vcm/chunk cc1
fix 1 all ave/time 100 1 100 c_myChunk[*] file tmp.out mode vector
```

3.169.4 Output info

This compute calculates a global array where the number of rows is the number of chunks *Nchunk* as calculated by the specified *compute chunk/atom* command. The number of columns is 3 for the (x, y, z) center-of-mass velocity coordinates of each chunk. These values can be accessed by any command that uses global array values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

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

3.169.5 Restrictions

none

3.169.6 Related commands

none

3.169.7 Default

none

3.170 compute viscosity/cos command

3.170.1 Syntax

```
compute ID group-ID viscosity/cos
```

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

3.170.2 Examples

```
units    real
compute  cos all viscosity/cos
variable V equal c_cos[7]
variable A equal 0.02E-5  # A/fs^2
variable density equal density
variable lz equal lz
variable reciprocalViscosity equal v_V/${A}/v_density*39.4784/v_lz/v_lz*100  # 1/(Pa*s)
```

3.170.3 Description

Define a computation that calculates the velocity amplitude of a group of atoms with an cosine-shaped velocity profile and the temperature of them after subtracting out the velocity profile before computing the kinetic energy. A compute of this style can be used by any command that computes a temperature (e.g., *thermo_modify*, *fix npt*).

This command together with *fix_accelerate/cos* enables viscosity calculation with periodic perturbation method, as described by *Hess*. An acceleration along the x-direction is applied to the simulation system by using *fix_accelerate/cos* command. 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 via

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

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

After the cosine-shaped collective velocity in the x-direction has been subtracted for each atom, the temperature is calculated by the formula

$$\mathrm{KE} = \frac{\mathrm{dim}}{2} N k_B T,$$

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

A symmetric tensor, stored as a six-element vector, is also calculated by this compute for use in the computation of a pressure tensor by the *compute pressue* command. The formula for the components of the tensor is the same as the above expression for E_{kin}, except that the 1/2 factor is NOT included and the v_i^2 is replaced by $v_{i,x}v_{i,y}$ for the xy component, and so on. Note that because it lacks the 1/2 factor, these tensor components are twice those of the traditional kinetic energy tensor. The six components of the vector are ordered xx, yy, zz, xy, xz, yz.

The number of atoms contributing to the temperature is assumed to be constant for the duration of the run; use the *dynamic* option of the *compute_modify* command if this is not the case. However, in order to get meaningful results, the group ID of this compute should be all.

The removal of the cosine-shaped velocity component by this command is essentially computing the temperature after a "bias" has been removed from the velocity of the atoms. If this compute is used with a fix command that performs thermostatting then this bias will be subtracted from each atom, thermostatting of the remaining thermal velocity will be performed, and the bias will be added back in. Thermostatting fixes that work in this way include *fix nvt*, *fix temp/rescale*, *fix temp/berendsen*, and *fix langevin*.

This compute subtracts out degrees of freedom due to fixes that constrain molecular motion, such as *fix shake* and *fix rigid*. This means that the temperature of groups of atoms that include these constraints will be computed correctly. If needed, the subtracted degrees of freedom can be altered using the *extra* option of the *compute_modify* command.

See the *Howto thermostat* page for a discussion of different ways to compute temperature and perform thermostatting.

3.170.4 Output info

This compute calculates a global scalar (the temperature) and a global vector of length 7, which can be accessed by indices 1–7. The first six elements of the vector are those of the symmetric tensor discussed above. The seventh is the cosine-shaped velocity amplitude V, which can be used to calculate the reciprocal viscosity, as shown in the example. These values can be used by any command that uses global scalar or vector values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

The scalar value calculated by this compute is "intensive". The first six elements of vector values are "extensive", and the seventh element of vector values is "intensive".

The scalar value is in temperature *units*. The first six elements of vector values are in energy *units*. The seventh element of vector value us in velocity *units*.

3.170.5 Restrictions

This compute 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 compute depends on *fix accelerate/cos* which can only work for 3d systems, it cannot be used for 2d systems.

3.170.6 Related commands

fix accelerate/cos

3.170.7 Default

none

(Hess) Hess, B. The Journal of Chemical Physics 2002, 116 (1), 209-217.

3.171 compute voronoi/atom command

3.171.1 Syntax

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

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

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

3.171.2 Examples

```
compute 1 all voronoi/atom
compute 2 precipitate voronoi/atom surface matrix
compute 3b precipitate voronoi/atom radius v_r
compute 4 solute voronoi/atom only_group
compute 5 defects voronoi/atom occupation
compute 6 all voronoi/atom neighbors yes
```

3.171.3 Description

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

Two per-atom quantities are calculated by this compute. The first is the volume of the Voronoi cell around each atom. Any point in an atom's Voronoi cell is closer to that atom than any other. The second is the number of faces of the Voronoi cell. This is equal to the number of nearest neighbors of the central atom, plus any exterior faces (see note below).

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

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

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

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

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

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

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

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

If the *occupation* keyword is specified the tessellation is only performed for the first invocation of the compute and then stored. For all following invocations of the compute the number of atoms in each Voronoi cell in the stored tessellation is counted. In this mode the compute returns a per-atom array with 2 columns. The first column is the number of atoms currently in the Voronoi volume defined by this atom at the time of the first invocation of the compute (note that the atom may have moved significantly). The second column contains the total number of atoms sharing the Voronoi cell of the stored tessellation at the location of the current atom. Numbers in column one can be any positive integer including zero, while column two values will always be greater than zero. Column one data can be used to locate vacancies (the coordinates are given by the atom coordinates at the time step when the compute was first invoked), while column two data can be used to identify interstitial atoms.

If the *neighbors* value is set to yes, then this compute also creates a local array with 3 columns. There is one row for each face of each Voronoi cell. The 3 columns are the atom ID of the atom that owns the cell, the atom ID of the atom in the neighboring cell (or zero if the face is external), and the area of the face. The array can be accessed by any command that uses local values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options. More specifically, the array can be accessed by a *dump local* command to write a file containing all the Voronoi neighbors in a system:

```
compute 6 all voronoi/atom neighbors yes
dump d2 all local 1 dump.neighbors index c_6[1] c_6[2] c_6[3]
```

If the *face_threshold* keyword is used, then only faces with areas greater than the threshold are stored.

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

Note

The calculation of Voronoi volumes is performed by each processor for the atoms it owns, and includes the effect of ghost atoms stored by the processor. This assumes that the Voronoi cells of owned atoms are not affected by atoms beyond the ghost atom cut-off distance. This is usually a good assumption for liquid and solid systems, but may lead to underestimation of Voronoi volumes in low density systems. By default, the set of ghost atoms stored by each processor is determined by the cutoff used for *pair_style* interactions. The cutoff can be set explicitly via the *comm_modify cutoff* command. The Voronoi cells for atoms adjacent to empty regions will extend into those regions up to the communication cutoff in x, y, or z. In that situation, an exterior face is created at the cutoff distance normal to the x, y, or z direction. For triclinic systems, the exterior face is parallel to the corresponding reciprocal lattice vector.

Note

The Voro++ package performs its calculation in 3d. This will still work for a 2d LAMMPS simulation, provided all the atoms have the same z-coordinate. The Voronoi cell of each atom will be a columnar polyhedron with constant cross-sectional area along the z-direction and two exterior faces at the top and bottom of the simulation box. If the atoms do not all have the same z-coordinate, then the columnar cells will be accordingly distorted. The cross-sectional area of each Voronoi cell can be obtained by dividing its volume by the z extent of the simulation box. Note that you define the z extent of the simulation box for 2d simulations when using the *create_box* or *read_data* commands.

3.171.4 Output info

Deprecated since version 21Nov2023: The *peratom* keyword was removed as it is no longer required.

This compute calculates a per-atom array with two columns. In regular dynamic tessellation mode the first column is the Voronoi volume, the second is the neighbor count, as described above (read above for the output data in case the *occupation* keyword is specified). These values can be accessed by any command that uses per-atom values from a compute as input. See the *Howto output* page for an overview of LAMMPS output options.

If the *edge_histo* keyword is used, then this compute generates a global vector of length *maxedge*+1, containing a histogram of the number of edges per face.

If the *neighbors* value is set to *yes*, then this compute calculates a local array with three columns. There is one row for each face of each Voronoi cell.

The Voronoi cell volume will be in distance *units* cubed. The Voronoi face area will be in distance *units* squared.

3.171.5 Restrictions

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

It also requires you have a copy of the Voro++ library built and installed on your system. See instructions on obtaining and installing the Voro++ software in the src/VORONOI/README file.

3.171.6 Related commands

dump custom, *dump local*

3.171.7 Default

The default for the neighbors keyword is no.

3.172 compute xrd command

3.172.1 Syntax

```
compute ID group-ID xrd lambda type1 type2 ... typeN keyword value ...
```

- ID, group-ID are documented in *compute* command
- xrd = style name of this compute command
- lambda = wavelength of incident radiation (length units)
- type1 type2 ... typeN = chemical symbol of each atom type (see valid options below)
- zero or more keyword/value pairs may be appended
- keyword = *2Theta* or *c* or *LP* or *manual* or *echo*

  2Theta values = Min2Theta Max2Theta
    Min2Theta,Max2Theta = minimum and maximum 2 theta range to explore
    (radians or degrees)
  c values = c1 c2 c3
    c1,c2,c3 = parameters to adjust the spacing of the reciprocal
               lattice nodes in the h, k, and l directions respectively
  LP value = switch to apply Lorentz-polarization factor
    0/1 = off/on
  manual = flag to use manual spacing of reciprocal lattice points
           based on the values of the c parameters
  echo = flag to provide extra output for debugging purposes

3.172.2 Examples

```
compute 1 all xrd 1.541838 Al O 2Theta 0.087 0.87 c 1 1 1 LP 1 echo
compute 2 all xrd 1.541838 Al O 2Theta 10 100 c 0.05 0.05 0.05 LP 1 manual

fix 1 all ave/histo/weight 1 1 1 0.087 0.87 250 c_1[1] c_1[2] mode vector file Rad2Theta.xrd
fix 2 all ave/histo/weight 1 1 1 10 100 250 c_2[1] c_2[2] mode vector file Deg2Theta.xrd
```

3.172.3 Description

Define a computation that calculates X-ray diffraction intensity as described in *(Coleman)* on a mesh of reciprocal lattice nodes defined by the entire simulation domain (or manually) using a simulated radiation of wavelength *lambda*.

The X-ray diffraction intensity, I, at each reciprocal lattice point, k, is computed from the structure factor, F, using the equations:

$$I = L_p(\theta)\frac{F^*F}{N}$$

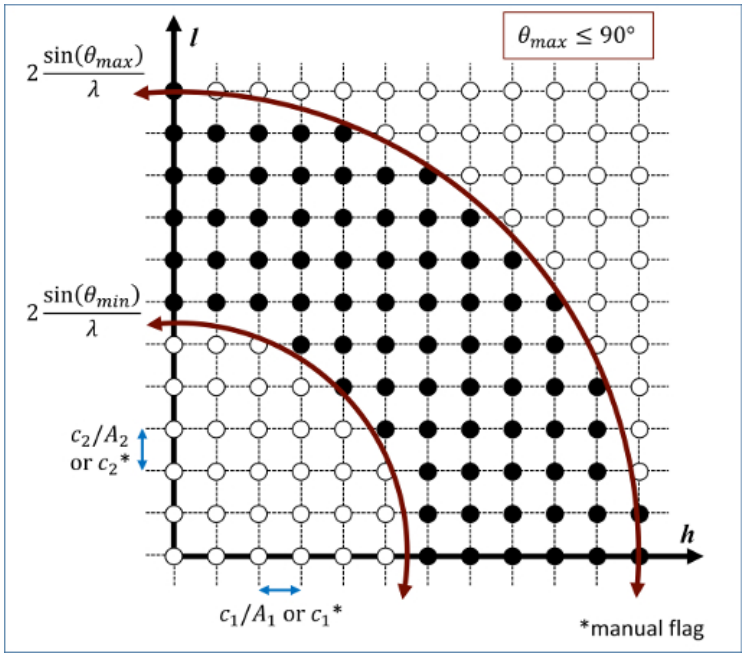
$$F(\mathbf{k}) = \sum_{j=1}^{N} f_j(\theta) exp(2\pi i \mathbf{k} \cdot \mathbf{r}_j)$$

$$L_p(\theta) = \frac{1+\cos^2(2\theta)}{\cos(\theta)\sin^2(\theta)}$$

$$\frac{\sin(\theta)}{\lambda} = \frac{\|\mathbf{k}\|}{2}$$

Here, $\mathbf{k}$ is the location of the reciprocal lattice node, r_j is the position of each atom, f_j are atomic scattering factors, Lp is the Lorentz-polarization factor, and θ is the scattering angle of diffraction. The Lorentz-polarization factor can be turned off using the optional *LP* keyword.

Diffraction intensities are calculated on a three-dimensional mesh of reciprocal lattice nodes. The mesh spacing is defined either (a) by the entire simulation domain or (b) manually using selected values as shown in the 2D diagram below.



For a mesh defined by the simulation domain, a rectilinear grid is constructed with spacing cA^{-1} along each reciprocal lattice axis, where A is a matrix containing the vectors corresponding to the edges of the simulation cell. If one or two directions has non-periodic boundary conditions, then the spacing in these directions is defined from the average of the (inversed) box lengths with periodic boundary conditions. Meshes defined by the simulation domain must contain at least one periodic boundary.

If the *manual* flag is included, the mesh of reciprocal lattice nodes will be defined using the c values for the spacing along each reciprocal lattice axis. Note that manual mapping of the reciprocal space mesh is good for comparing diffraction results from multiple simulations; however, it can reduce the likelihood that Bragg reflections will be satisfied unless

small spacing parameters (< 0.05 Å^{-1}) are implemented. Meshes with manual spacing do not require a periodic boundary.

The limits of the reciprocal lattice mesh are determined by range of scattering angles explored. The *2Theta* parameter allows the user to reduce the scattering angle range to only the region of interest which reduces the cost of the computation.

The atomic scattering factor, f_j, accounts for the reduction in diffraction intensity due to Compton scattering. Compute xrd uses analytical approximations of the atomic scattering factors that vary for each atom type (type1 type2 ... typeN) and angle of diffraction. The analytic approximation is computed using the formula *(Colliex)*:

$$f_j\left(\frac{\sin(\theta)}{\lambda}\right) = \sum_{i=1}^{4} a_i \exp\left(-b_i \frac{\sin^2(\theta)}{\lambda^2}\right) + c$$

Coefficients parameterized by *(Peng)* are assigned for each atom type designating the chemical symbol and charge of each atom type. Valid chemical symbols for compute xrd are:

H	He1-	He	Li	Li1+
Be	Be2+	B	C	Cval
N	O	O1-	F	F1-
Ne	Na	Na1+	Mg	Mg2+
Al	Al3+	Si	Sival	Si4+
P	S	Cl	Cl1-	Ar
K	Ca	Ca2+	Sc	Sc3+
Ti	Ti2+	Ti3+	Ti4+	V
V2+	V3+	V5+	Cr	Cr2+
Cr3+	Mn	Mn2+	Mn3+	Mn4+
Fe	Fe2+	Fe3+	Co	Co2+
Co	Ni	Ni2+	Ni3+	Cu
Cu1+	Cu2+	Zn	Zn2+	Ga
Ga3+	Ge	Ge4+	As	Se
Br	Br1-	Kr	Rb	Rb1+
Sr	Sr2+	Y	Y3+	Zr
Zr4+	Nb	Nb3+	Nb5+	Mo
Mo3+	Mo5+	Mo6+	Tc	Ru
Ru3+	Ru4+	Rh	Rh3+	Rh4+
Pd	Pd2+	Pd4+	Ag	Ag1+
Ag2+	Cd	Cd2+	In	In3+
Sn	Sn2+	Sn4+	Sb	Sb3+
Sb5+	Te	I	I1-	Xe
Cs	Cs1+	Ba	Ba2+	La
La3+	Ce	Ce3+	Ce4+	Pr
Pr3+	Pr4+	Nd	Nd3+	Pm
Pm3+	Sm	Sm3+	Eu	Eu2+
Eu3+	Gd	Gd3+	Tb	Tb3+
Dy	Dy3+	Ho	Ho3+	Er
Er3+	Tm	Tm3+	Yb	Yb2+
Yb3+	Lu	Lu3+	Hf	Hf4+
Ta	Ta5+	W	W6+	Re
Os	Os4+	Ir	Ir3+	Ir4+
Pt	Pt2+	Pt4+	Au	Au1+
Au3+	Hg	Hg1+	Hg2+	Tl
Tl1+	Tl3+	Pb	Pb2+	Pb4+

continues on next page

Table 1 – continued from previous page

Bi	Bi3+	Bi5+	Po	At
Rn	Fr	Ra	Ra2+	Ac
Ac3+	Th	Th4+	Pa	U
U3+	U4+	U6+	Np	Np3+
Np4+	Np6+	Pu	Pu3+	Pu4+
Pu6+	Am	Cm	Bk	Cf

If the *echo* keyword is specified, compute xrd will provide extra reporting information to the screen.

3.172.4 Output info

This compute calculates a global array. The number of rows in the array is the number of reciprocal lattice nodes that are explored which by the mesh. The global array has two columns.

The first column contains the diffraction angle in the units (radians or degrees) provided with the *2Theta* values. The second column contains the computed diffraction intensities as described above.

The array can be accessed by any command that uses global values from a compute as input. See the *Howto output* doc page for an overview of LAMMPS output options.

All array values calculated by this compute are "intensive".

3.172.5 Restrictions

This compute 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.

The compute_xrd command does not work for triclinic cells.

3.172.6 Related commands

fix ave/histo, *compute saed*

3.172.7 Default

The option defaults are *2Theta* = 1 179 (degrees), *c* = 1 1 1, *LP* = 1, no manual flag, no echo flag.

(Coleman) Coleman, Spearot, Capolungo, MSMSE, 21, 055020 (2013).

(Colliex) Colliex et al. International Tables for Crystallography Volume C: Mathematical and Chemical Tables, 249-429 (2004).

(Peng) Peng, Ren, Dudarev, Whelan, Acta Crystallogr. A, 52, 257-76 (1996).