Particle Dynamics Parallel Simulator (PDPS)
Command Manual

Lingqi Yang [1] Huiming Yin
Civil Engineering and Engineering Mechanics, Columbia University

October 17, 2014

[1] ly2282@columbia.edu

Preface

Particle Dynamics Parallel Simulator (PDPS) is a software package developed by Sustainable Engineering and Materials Laboratory (SEML) from Columbia University. It is inspired and designed based on the structure from the open source software LAMMPS (http://lammps.sandia.gov/index.html) developed at Sandia National Laboratory. It brought our interest to develop this new software package when observing the lack of available source to conduct particle dynamics simulation with various of particle based potentials, such as DPD, DEM, etc. It is our motivation to make an example and provide a general platform for researchers interested in this area to refer and explore. With the supervision and support from Prof. Huiming Yin, Lingqi Yang as the main designer and developer, SEML group aims to run simulation of particle-based method which bridges the atomic scale and meso-scale for investigating material behavior with various research purposes. It inherits the spirit of molecular dynamics method algorithm which is based on Newton's second law. However, this open source code has been designed for particle dynamics simulation with a tunable time and length scale. It can be used in simulating the dynamics and rheological properties of simple and complex fluids, or granular materials. It is our intention to create a parallel open source code that is easy and convenient for researchers to modify and extend to meet their own research needs.

PDPS is currently written in C++. The parallelism is fulfilled by using the domain decomposition technique and managed by the message passing interface (MPI) library. It can be run on single processor or multiple processors machine which can compile C++ and support MPI library. It is a free open source software package under the GNU license.

In this document, the instruction of each command is provided in terms of category and alphabetic order. Currently, we are still adding more details into this manual. For any unclear command instruction, user can follow LAMMPS website for more details. If you have any question, please feel free to contact the author for further information.

Contents

Chapter 1

Getting Started

1.1 Download

The latest version of PDPS can be downloaded at http://www.columbia.edu/cu/civileng/yin/software.html . The latest command manual is also provided in a pdf version.

PDPS - Beta 0.6: Link

1.2 Prepare Input Script

The interface of PDPS software package is by preparing an input script file, where user can add command line by line. Comment line will be started with ♯ and empty line will be skipped. It is a pretty much free style for user to configure the simulation as he or she prefers. However, certain rules are illustrated in the Sec. 2. Some commands have to be after some commands. Otherwise, error message will be printed to the screen.

Generally speaking, the oder of commands can be explained by four steps:

(1) Setup simulation fundamental parameters, such as timestep, units, dimension, boundary conditions.

(2) Create simulation box, region in order to create particle and assign them to a specific group.

(3) Set potential coefficients, add the correct ensemble, necessary constraints, properties to be computed and output

(4) Setup output file and run the simulation

Below, the input script file for pure water will be presented as the very example:

From the above input script, it can be seen that the first part has defined some fundamental parameters for the simulation environment. It is a 3D simulation, periodic boundary conditions are applied along all

```
# pure warter simulation by PDPS source code

# Initial conidtions
dimension           3
boundary                        p p p
lattice             spacing x 0.5 y 0.5 z 1
units               lj
timestep            0.04
neighbor            0.3
neigh_modify        every 1 delay 1 check yes


# Create box
region              box block 0 10 0 10 0 10
create_box          1 box


# Set mass
mass                1 1

create_particle         1 region box

group               all

pair_style          dpd 1.0 1.0 3442527
pair_coeff          * * 25.0 4.5 1.0

velocity            all create 10.0 10342 gaussian

compute             1 all temp
compute             2 all pressure

integrate           v_verlet

fix                 1 all nve

thermo              1
thermo_style        custom step temp press

dump                1 all atom 1 dump.atom

run                             200
```

directions, uniform lattice spacing are set along all directions, "lj" units are used, timestep is set as 0.04 and neighbor list will be rebuilt at each step.

The second part will create box based on the defined region and particles with type 1 will be created uniformly in the simulation box. Total number of particle is 4000 and the mass of each particle is defined as 1.

In the third part, DPD potential is chosen and coefficients are set for all possible pairs. Initial velocity with Gaussian distribution will be created at temperature 10. Two properties, temperature and pressure, will be computed.

In the last stage, computed temperature and pressure will be written to a log file every 1 step during the simulation, and particles information, coordinates or velocities, will be output to a dump file named "dump.atom" with "atom" format so that it can be visualized by VMD software. Simulation will be run for 200 steps.

In a summary, it is user's responsibility to create a reasonable environment and parameters for conducting the simulation. This procedure is quite similar as the real experiment, such as preparing sample, setup equipment, run the test. It is highly suggested to check each command's instruction in details before running a efficient and correct simulation.

1.3 Submit job

In the Windows platform, one needs to install the Microsoft high performance computing (HPC) to include "mpi.h" if PDPS is run in Visual studio. After compiling the code, one can either run it sequentially in the Visual Studio, or open a command prompt in Windows to type the following command:

```
mpiexec -n 4 Debug/pdps.exe
```

A short instruction to compile and run MPI program can be found in http://blogs.msdn.com/b/risman/archive/2009/01/04/ms-mpi-with-visual-studio-2008.aspx

To compile the code in Linux system, one can use the following command:

```
mpiCC -O2 *.h *.cpp
```

To run the code in Linux system, normally one needs to setup a PBS batch file to run it. Since it varies from platform, it is suggested to contact with your local IT technician. For your convenience, an example of such file (run.pdps) is given below:

```
#PBS -l nodes=1:ppn=8
#PBS -l walltime=48:00:00,mem=2gb
#PBS -W group_list=yetiastro
#PBS -N boundary_f

cd $PBS_O_WORKDIR

mpirun -np 8 ps_yeti liquid.in > screen
```

By typing the command "qsub run.pdps", the job will be submitted to the cluster.

Chapter 2

Command Instruction

Commands are divided into both category and alphabetic order. In the category, the commands can be found by its function.

2.1 Commands by category

Commands by category.

2.1.1 Initialization

boundary, dimension, neighbor neighbor_modify, processors, timestep, units

2.1.2 Geometry definition

create_box, region

2.1.3 Particle definition

particle_style, create_particle, mass, read_data, group

2.1.4 Particle interaction

pair_style, pair_coeff

2.1.5 Analysis and constraint

analyze unanalyze compute, uncompute, fix, unfix

2.1.6 Simulation procedure

run, reset_timestep

2.1.7 Output

thermo, thermo_style, dump, undump

2.2 Commands by alphabetic

Commands by alphabetic order.

a-n

analyze boundary, compute, create_box, create_particle, dimension, dump, fix, group, mass, neighbor, neighbor_modify

o-r

pair_coeff, pair_style, particle_style, processors, read_data, region, reset_timestep, run

r-z

timestep, thermo, thermo_style, unanalyze, uncompute, undump, unfix, units

2.2.1 analyze

analyze is used to analyze data computed by other class.

Syntax:

```
analyze          id group-id style args
```

- id: user assigned name for this *analyze*
- group-id: only particles in this group are applied with this *analyze*
- style: type of the *analyze* (see the list below)
- args: arguments to the corresponding *analyze style*

Examples:

```
analyze          1 Al ave/space 1 1 1000 z center 1 density/number file
   vol_Al.txt
analyze          3 all ave/time 1 10000 10000 c_1 file Al-Al.rdf
```

Description:

A list of current available *fix style* is given below:

ave/space	perform time average on spatial average
ave/time	perform time average

Table 2.1: List of available *analyze style*

2.2.1.1 analyze ave/space

analyze ave/space is used to perform both time and spatial average on selected data.

Syntax:

```
analyze          id group-id ave/space nevery nrepeat nfreq dimension
   location options-keyword
```

- id: user assigned name for this *fix*
- group-id: the *fix* is applied only to the particles in this group id
- nevery, nrepeat, nfreq: adjust sample size for the spatial average
- dimension: x, y or z
- location: *lower*, *center* or *upper*
- options-keyword:

  density/number

  density/mass

  file file_name

Examples:

```
analyze          1 Al ave/space 1 1 1000 z center 1 density/number file
   vol_Al.txt
```

Description:

If option-keyword *density/number* is used, the space average is performed on the number of density.

If option-keyword *density/mass* is used, the space average is performed on the mass density.

If option-keyword *file* is used, the space average results are output to the file.

2.2.1.2 analyze ave/time

analyze ave/time is used to perform time average on selected data.

Syntax:

```
analyze          id group-id ave/time nevery nrepeat nfreq c_compute-id
   options-keyword
```

- id: user assigned name for this *fix*
- group-id: the *fix* is applied only to the particles in this group id
- nevery, nrepeat, nfreq: adjust sample size for the spatial average
- c_compute-id: select computed data
- options-keyword:

  file file_name

Examples:

```
analyze          3 all ave/time 1 10000 10000 c_1 file Al-Al.rdf
```

Description:

The compute-id is made of "c_compute-id", eg. *c_1*, *c_temp*.

If option-keyword *file* is used, the time average results are output to the file.

2.2.2 boundary

Define the boundary condition of the simulation box.

Syntax:

```
boundary                 x y z
```

- x, y, z: *p* or *f*

Examples:

```
boundary                 p p f
```

Description:

p means periodic boundary condition and *f* means a fixed boundary condition. If a particle crosses a fixed boundary condition, it will be lost. The program will be terminated and an error message will be shown.

2.2.3 compute

compute is used to compute certain data specified by the user.

Syntax:

```
compute          id group-id style args
```

- id: user assigned name for this *compute*
- group-id: only particles in this group are applied with this *compute*
- style: type of the *compute* (see the list below)
- args: arguments to the corresponding *compute style*

Examples:

```
compute          1 all centroid
compute          2 water temp
```

Description:

A list of current available *compute style* is given below:

centroid	compute centroid of a group of particles
pressure	perform time average
rdf	compute radial distribution of a group of particles
temp	perform time average on spatial average

Table 2.2: List of available compute style

Output

The output can be in the form of scalar, vector or array.

- scalar: Results can be accessed by c_id, where id is the compute-id, eg. *c_1*
- vector: Results can be accessed by c_id[dim], where dim is 0, 1, ... N, eg. *c_1[0]*
- array: Results can be accessed by c_id[dim1][dim2], *c_2[0][2]*

2.2.3.1 compute centroid

compute centroid is used to calculate the centroid of a group of particles.

Syntax:

```
compute          id group-id centroid
```

- id: user assigned name for this *fix*
- group-id: the *fix* is applied only to the particles in this group id

Examples:

```
compute          5 all centroid
```

Description:

This compute will calculate the centroid of a group of particles along x, y and z coordinates:

$$x_c = \frac{1}{N}\sum_{i=1}^{N} x_i$$

where N is the total number of particles in the group.

Output:

The output is in the form of array. Results can be accessed by *c_id[dim]* in the *thermo_modify custom* command, where *id* is the compute id and dim ranges from 0 to 2, eg. *c_*5[0].

2.2.3.2 compute pressure

compute pressure is used to calculate the pressure of the whole system.

Syntax:

```
compute          id group-id pressure
```

- id: user assigned name for this *fix*
- group-id: the group-id has to be *all*

Examples:

```
compute          3 all pressure
```

Description:

This compute will calculate the pressure of the whole system by using Virial theorem.

Output:

The output is in the form of scalar. Results can be accessed by *c_id* in the *thermo_modify custom* command, where *id* is the compute id.

2.2.3.3 compute rdf

compute rdf is used to calculate the radial distribution function of a specified type of particle.

Syntax:

```
compute          id group-id rdf nevery i-type j-type
```

- id: user assigned name for this *fix*
- group-id: only particles in this group are applied with this *compute*

- i-type: center particle type for calculating rdf
- j-type: surrounding particle type for calculating rdf

Examples:

```
compute          1 all rdf 100 1 1
```

Description:

This compute calculates the radial distribution of j-type particles as a function of i-type particles' radius. Both i-type and j-type particles are in the same group specified by the group-id.

Output:

The output is in the form of array: radius distance r, $g(r)$ and accumulated particles at distance r.

2.2.3.4 compute temp

compute temp is used to calculate the temperature of a group of particle.

Syntax:

```
compute          id group-id temp
```

- id: user assigned name for this *fix*
- group-id: only particles in this group are applied with this *compute*

Examples:

```
compute          1 all temp
```

Description:

This compute calculates the temperature of a group of particles based on the kinetic theory of gases. The expression of temperature is given as:

$$T = \langle \mathbf{v}^2 \rangle / 3k_B$$

Output:

The output is in the form of scalar.

2.2.4 create_box

create_box is used to define the dimension of the simulation box.

Syntax:

```
create_box              ntypes region-id
```

- ntypes: total number of types of particles to be created
- region-id: id of region created by *region* command

Examples:

```
create_box              2 box
```

Description:

The geometry of the simulation box is defined by the *region* command.

Restriction:

A simulation box has to be created before creating particles.

2.2.5 create_particle

create_particle is used to define the dimension of the simulation box.

Syntax:

```
create_particle         type style args
```

- type: only this type of particles to be created
- style: type of the *create_particle* command
  - *random_no_overlap*: microstructure generator for creating particles with no overlap

    N region-id cell-length random-seed
  - *single*: create single particle by coordinates

    x y z
  - *spacing*: create particle uniformly in a specified region

    region-id: id of the region created by *region* command

Examples:

```
create_particle         1 random_no_overlap 100 box1 0.5 23432
create_particle         2 single 0.5 0.5 1.0
```

Description:

If the style *random_no_overlap* is used, sphere particles are created randomly over the region domain and no overlap is allowed. If the style *spacing* is used, particles are generated uniformly and the lattice spacing is defined by the *lattice* command. If the style *single* is used, only one particle is created based on the user-defined coordinates.

Restriction:

It has to be after the simulation box is defined.

2.2.6 dimension

Define the dimension of the simulation box.

Syntax:

```
dimension                N
```

- N: dimension can be 2 or 3

Examples:

```
dimension                3
```

Description:

It defines the problem as 3D or 2D.

Default:

N = 3

2.2.7 dump

dump is used to output particle related information.

Syntax:

```
dump             id group-id style args
```

- id: user assigned name for this *dump*
- group-id: only particles in this group are output with this *dump*
- style: style of the *dump* (see the list below)
- args: arguments to the corresponding *dump style*

Examples:

```
dump            1 all atom 50 dump.atom
dump            2 Al custom 10 a18.75.txt step x y z vx vy vz
```

Description:

This command is used to write particle related information, such as coordinate, velocity, etc., into specified file. A list of current available *compute style* is given below:

atom	compute centroid of a group of particles
custom	perform time average

Table 2.3: List of available dump style

2.2.7.1 dump atom

dump atom is used to write particle related information in the style of *atom*.

Syntax:

```
dump            id group-id atom nevery file-name
```

- id: user assigned name for this *fix*
- group-id: the *fix* is applied only to the particles in this group id
- nevery: output frequency
- file-name: name of the file

Examples:

```
dump            1 all atom 50 dump.atom
```

Description:

An example of the *atom* style dump is given below:

```
ITEM: TIMESTEP
0
ITEM: NUMBER OF ATOMS
5
ITEM: BOX BOUNDS
0 10
0 10
-0.6 9.6
ITEM: ATOMS id type xs ys zs
0 1 0.01 0.01 0.003
1 1 1.01 0.01 0.003
2 1 2.01 0.01 0.003
```

```
3 1 3.01 0.01 0.003
4 1 4.01 0.01 0.003
5 1 5.01 0.01 0.003
ITEM: TIMESTEP
50
ITEM: NUMBER OF ATOMS
5
ITEM: BOX BOUNDS
0 10
0 10
-0.6 9.6
ITEM: ATOMS id type xs ys zs
0 1 0.02 0.012 0.013
1 1 1.02 0.11 0.011
2 1 2.11 0.09 0.001
3 1 3.02 0.03 0.002
4 1 4.21 0.21 0.004
5 1 5.11 0.21 0.013
⋮ ⋮ ⋮ ⋮ ⋮
```

2.2.7.2 dump custom

dump custom is used to calculate the centroid of a group of particles.

Syntax:

```
dump             id group-id custom nevery file-name options-keyword
```

- id: user assigned name for this *fix*
- group-id: the *fix* is applied only to the particles in this group id
- nevery: output frequency
- file-name: name of the file
- options-keyword:

  id: particle global id

  step: time step

  type: particle type

  x, y, z: particle coordinate

  vx, vy, vz: particle velocity

  fx, fy, fz: particle force

Examples:

```
dump             2 Al custom 1 a18.75.txt step x y z vx vy vz
```

Description:

This command will dump user-defined information into the specified file.

2.2.8 fix

fix is used to add constraints or additional conditions to the simulation.

Syntax:

```
fix                         id group-id style args
```

- id: user assigned name for this *fix*
- group-id: the *fix* is applied only to the particles in this group id
- style: type of the *fix* (see the list below)
- args: arguments to the corresponding *fix style*

Examples:

```
fix                         1 all nve
fix                         2 Al acc gravity z -0.592113
```

Description:

A list of current available *fix style* is given below:

acc	add acceleration
addforce	add a specified force to a certain group of particles
nve	the NVE ensemble
setforce	set a constant force to a certain group of particles
wall/dem	set a constant force to a certain group of particles
wall/reflect	applied certain rules to particles across the user-defined wall

Table 2.4: List of available *fix style*

2.2.8.1 fix acc

fix acc is used to add user-specified acceleration to a certain group of particles.

Syntax:

```
fix                         id group-id acc options-keyword
```

- id: user assigned name for this *fix*
- group-id: the *fix* is applied only to the particles in this group id
- options-keyword: *gravity*, *region*

  gravity dimension value

region region-id

Examples:

```
fix                     2 Al acc gravity z -0.592113 region box2
```

Description:

If option-keyword *gravity* is used, the user-defined magnitude ("value") can only be applied in one "dimension", which can be specified by x, y or z. A negative value will indicate an opposite direction of standard Cartesian coordinate.

If option-keyword *region* is used, only particles in this "region id" will be applied with this fix.

2.2.8.2 fix addforce

fix addforce is used to add user-specified force to a certain group of particles.

Syntax:

```
fix                     id group-id addforce options-keyword
```

- id: user assigned name for this *fix*
- group-id: the *fix* is applied only to the particles in this group id
- options-keyword: *drag_stokes*, *buoyancy*

  drag_stokes μ

  buoyancy dimension ρ g

Examples:

```
fix                     drag water addforce drag_stokes 1.0950
fix                     floating water addforce buoyancy z 0.789 9.81
```

Description:

If option-keyword *drag_stokes* is used, the user-defined viscosity (μ) will be used to calculate the drag force added on particles in the specified group based on Stokes' law:

$$F_d = 6\pi\mu Rv$$

where R and v are particle's radius and velocity, respectively.

If option-keyword *buoyancy* is used, the buoyancy force will be added to particles in the specified group along x, y or z "dimension":

$$F_b = \rho g \frac{4}{3}\pi R^3$$

where ρ is the density of the liquid and g stands for the user-defined gravity. A negative value of g will indicate an opposite direction of standard Cartesian coordinate.

2.2.8.3 fix nve

fix nve is used to add user-specified acceleration to a certain group of particles.

Syntax:

```
fix                    id group-id nve
```

- id: user assigned name for this *fix*
- group-id: the *fix* is applied only to the particles in this group id

Examples:

```
fix                    1 all nve
```

Description:

This fix will perform *nve* ensemble time integration.

2.2.8.4 fix setforce

Set a group of particles' force components as exact numerical values.

Syntax:

```
fix              id group-id setforce options-keyword
```

- id: user assigned name for this *fix*
- group-id: the *fix* is applied only to the particles in this group id
- options-keyword:

  fx, fy, fz: force components

Examples:

```
fix              3 freeze setforce 0.0 0.0 0.0
```

Description:

This fix will set a group of particles' force components as exact numerical values.

2.2.8.5 fix wall/dem

Create wall and define wall-particle interaction as discrete element method (DEM).

Syntax:

```
fix              id group-id wall/dem options-keyword e kn Cn kt Ct μ cut
   -off
```

- id: user assigned name for this *fix*
- group-id: the *fix* is applied only to the particles in this group id
- options-keyword: *region*

  region region-id
- e, kn Cn kt Ct μ cut-off: dem force parameters

Examples:

```
fix              2 all wall/dem lsd region wall1 region wall2 0.85 3000
   28.6527 571.43 15.82634 0.4 0.5
```

Description:

This fix will create wall and the wall-particle interaction is defined as DEM.

2.2.8.6 fix wall/reflect

Create wall and apply reflection rule when particle cross the wall.

Syntax:

```
fix              id group-id wall/reflect style options-keyword
```

- id: user assigned name for this *fix*
- group-id: the *fix* is applied only to the particles in this group id
- style: *bounce_back*, *bounce_forward*, *specular*
- options-keyword: *xlo*, *xhi*, *ylo*, *yhi*, *zlo*, *zhi*

  xlo value

Examples:

```
fix              3 mobile wall/reflect style xlo 0 xhi 40 ylo 0 yhi 40
```

Description:

If the style *bounce_back* is used, the position of the particle crossing the wall will be reversed and the direction of its velocity and force components are reversed.

If the style *bounce_forward* is used, the position of the particle crossing the wall will be specular reflected and the direction of its velocity and force components are reversed.

If the style *specular* is used, the position of the particle crossing the wall will be specular reflected and only the normal direction of its velocity and force components will be reversed.

2.2.9 group

Assign particle to a group

Syntax:

```
group            id style
```

- style: *all*, *region*, *type*

  region region-id

  type particle-type

Examples:

```
group            all
group            g1 region top
group            Al type 1
```

Description:

If the style *region* is used, particle within this region at the beginning will be assigned to this group.

If the style *type* is used, particle of this type will be assigned to this group.

Before every run command, particle list in this group will be updated.

Default:

Whether group *all* is specified in the input script or not, the first group is always *all* which means all particles belong to the first group.

Restriction:

The maximum number of group can be created is 31.

2.2.10 integrate

Assign particle to a group

Syntax:

```
integrate                style
```

- style: *v_verlet*

  v_verlet: velocity verlet integration method

Examples:

```
integrate                v_verlet
```

Description:

If the style *v_verlet* is used, the velocity Verlet algorithm is used.

2.2.11 mass

Assign mass to a type of particle.

Syntax:

```
mass             type value
```

- type: particle-type
- value: mass-value

Examples:

```
mass             1 0.5
```

Description:

All type of particles' mass has to be defined.

2.2.12 neighbor

Setup neighbor list.

Syntax:

```
neighbor                 skin style
```

- skin: skin distance
- style: *single*, *multi*

Examples:

```
neighbor                 0.5 single
neighbor                 0.5 multi
```

Description:

single is used when cut-off of different particle-particle interaction is in a short range.

multi is used when cut-off is in a long range.

2.2.13 neighbor_modify

Setup neighbor list.

Syntax:

```
neigh_modify              options-keyword
```

- options-keyword: *delay*, *every*, *check*

  delay value

  every value

  check yes, no

Examples:

```
neigh_modify              delay 1 every 1 check yes
```

Description:

The frequency to build the neighbor list is controlled by the *delay*, *every* and *check* option keywords. The neighbor list is only built at least *delay* steps after the previous build. Then, every *every* steps the neighbor list will be built after the *delay*. If *check yes* is specified, the program will check the maximum displacement of particles every *every* steps after the *delay* from the last build. Only if the maximum displacement is over half of the $cut-off+skin$, the neighbor list will be rebuilt. If *check no* is specified, the neighbor list will be rebuilt after the *delay*, even with *every* specified.

2.2.14 pair_coeff

Setup pair coefficients for a pair style specified in pair_style command.

Syntax:

```
pair_coeff                i-type j-type args
```

- i-type, j-type: interaction between i-type and j-type particles

Examples:

```
pair_style                dem/lsd 0.3 1.0
pair_coeff                * * 1000 16.6016 314.29 6.3008 0.5 1
```

Description:

If *i-type* or *j-type* is set as *, it ranges from all types of particles. The coefficients format is related to the pair style which can be found in pair_style.

2.2.15 pair_style

Setup pair style.

Syntax:

```
pair_coeff              i-type j-type args
```

- i-type, j-type: interaction between i-type and j-type particles

Examples:

```
pair_style              dem/lsd 0.3 1.0
pair_coeff              * * 1000 16.6016 314.29 6.3008 0.5 1
```

Description:

A list of current available *pair style* is given below:

pair_style dem/lsd	add acceleration
pair_style dpd	add a specified force to a certain group of particles

Table 2.5: List of available *pair style*

2.2.15.1 pair_style dem/lsd

Setup the discrete element method (DEM) linear dash-pot (lsd) pair style.

```
pair_style              dem/lsd e r_gcut
pair_coeff              i-type j-type kn Cn kt Ct e r_cut
```

- temp: target temperature
- r_{gcut}: global cut-off
- random-seed: seed for generating random number
- kn, Cn, kt, Ct, e, r_{cut}: DEM force parameters

Examples:

```
pair_style              dem/lsd 0.3 1.0
pair_coeff              * * 1000 16.6016 314.29 6.3008 0.5 1
```

Description:

The force in the lsd DEM potential have spring and tangential forces along both normal and tangential directions. The tangential force is under the Coulomb friction rule.

$$\boldsymbol{f}_n^{ij} = k_n \Delta \boldsymbol{r}_n^{ij} - C_n \boldsymbol{v}_n^{ij}$$
$$\boldsymbol{f}_t^{ij} = min(\mu |\boldsymbol{f}_n^{ij}|, |-k_t \Delta \boldsymbol{r}_t^{ij} - C_t \boldsymbol{v}_t^{ij}|)\boldsymbol{t}^{ij}$$

2.2.15.2 pair_style dpd

Setup the dissipative particle dynamics (DPD) pair style.

```
pair_style              dpd temp r_gcut random-seed
pair_coeff              i-type j-type a γ r_cut
```

- temp: target temperature
- r_{gcut}: global cut-off
- random-seed: seed for generating random number
- a, γ, r_{cut}: DPD force parameters

Examples:

```
pair_style              dpd 1.0 1.0 22717 2
pair_coeff              * * 18.75 4.5 1.0
```

Description:

The force in the DPD potential have three components: conservative force, dissipative force and random force:

$$\boldsymbol{f}^I = \sum_{J \neq I} (\boldsymbol{f}_C^{IJ} + \boldsymbol{f}_D^{IJ} + \boldsymbol{f}_R^{IJ}).$$

The formulation of these three force components are given below:

$$\begin{aligned} \boldsymbol{f}_C^{IJ} &= a^{IJ} \omega_C \boldsymbol{n}^{IJ} \\ \boldsymbol{f}_D^{IJ} &= -\gamma \omega_D (\boldsymbol{n}^{IJ} \cdot \boldsymbol{v}^{IJ}) \boldsymbol{n}^{IJ} \\ \boldsymbol{f}_R^{IJ} &= \sigma \omega_R \zeta^{IJ} (\triangle t)^{-1/2} \boldsymbol{n}^{IJ} \end{aligned}$$

where $\omega_D = (\omega_R)^2 =$, $\sigma = \sqrt{2 k_B T \gamma}$, and r_{cut} is the local cutoff set by the command pair_coeff. ζ^{IJ} is the random number and the "random-seed" is used to generate random number. In DPD, the random number is generated with a Gaussian distribution.

A common choice for the weight functions are given as

$$\omega_C = \begin{cases} 1 - \dfrac{r^{IJ}}{r_{\text{cut}}} & (r^{IJ} < r_{\text{cut}}), \\ 0 & (r^{IJ} \geq r_{\text{cut}}), \end{cases}$$

$$\begin{aligned} \omega_D(r^{IJ}) &= [\omega_R(r^{IJ})]^2 \\ &= \begin{cases} \left(1 - \dfrac{r^{IJ}}{r_{\text{cut}}}\right)^2 & (r^{IJ} < r_{\text{cut}}), \\ 0 & (r^{IJ} \geq r_{\text{cut}}), \end{cases} \end{aligned}$$

style	data structure
atomic	$x[3]$, $v[3]$, $f[3]$
sphere	$x[3]$, $v[3]$, $f[3]$, $\omega[3]$, $torque[3]$

Table 2.6: Data structure of particle style

2.2.16 particle_style

Define the style of particle

Syntax:

```
particle                style
```

- style: *atomic*, *sphere*

Examples:

```
particle                sphere
```

Description:

Particle style defines the data structure of the particle. A summary of the data structure of the particle style is given below:

Default

style = *atomic*

2.2.17 processors

Define how the processors distributed after the domain decomposition.

Syntax:

```
processors              nx ny nz
```

- nx, ny, nz: processor grid

Examples:

```
processors              4 4 1
```

Description:

The processor grid determines how the working load is distributed among processors.

Suggestion

If the particles fill the entire domain relatively uniformly, normally, there is no need to specify how to determine the processor grid. However, if not, the user can specify the processor grid for the sake of best efficiency.

Restriction

$nx * ny * nz$ has to be equal to the total required number of processors.

Default

The processor grid is determined by the minimal surface area of sub-domain.

2.2.18 read_data

Read particle information from a data file.

Syntax:

```
read_data                file-name
```

- file-name: name of the file

Examples:

```
read_data                initial_structure.data
```

Description:

This command is used to read particle information data from a file. Currently, the file's format follows the LAMMPS format.

Suggestion

If PDPS cannot generate user required structure, one can use LAMMPS to generate the desired structure and use this command to read it.

2.2.19 region

User-defined geometry.

Syntax:

```
region           id style args
```

- id: user assigned name for this region
- style: *block*, *polygon*

- *block* xlo xhi ylo yhi zlo zhi
- *polygon* pt_1 pt_2 pt_3 ...

Examples:

```
region          box block 0 25 0 10 0 25
region          wall1 polygon p1 15 0 0 p2 15 10 0 p3 10 10 0 p4 10 0 0
```

Description:

If the style *block* is used, xlo, ylo, and zlo will define the lower bound of the simulation box.

If the style *polygon* is used, user needs to construct the polygon one point by one point. The sequence of the firs three points determines the normal direction of the surface:

$$\boldsymbol{n} = \frac{(\boldsymbol{pt}_2 - \boldsymbol{pt}_1) \times (\boldsymbol{pt}_3 - \boldsymbol{pt}_1)}{|(\boldsymbol{pt}_2 - \boldsymbol{pt}_1) \times (\boldsymbol{pt}_3 - \boldsymbol{pt}_1)|}$$

Restriction:

Reset the time step.

2.2.20 reset_timestep

Define the boundary condition of the simulation box.

Syntax:

```
reset_timestep          value
```

- value: time step to be reset to

Examples:

```
reset_timestep          0
```

Suggestion:

Reset the time step when some commands are invoked and needs the time step to be reset. For example, when *analyze ave/time* is used, current time step may have effect on the time averaging frequency.

2.2.21 run

Define the simulation duration.

Syntax:

```
run                value
```

- value: simulation duration

Examples:

```
run                10000
```

Description:

Multiple run can be set in the input script.

Suggestion For example, a first run can be set for the thermal equilibrium and a second run can start the simulation.

2.2.22 thermo

Define the log file data output frequency.

Syntax:

```
thermo             value
```

- value: log file data output frequency

Examples:

```
thermo             10
```

Description:

The log file will always write data before the run, which means if the simulation duration is 10 steps and *thermo 1* is used, data will be written into the log file for 11 times.

2.2.23 thermo_style

Define which data to be written in the log file.

Syntax:

```
thermo_style              style args
```

- style: *custom*
- args: *step*, *press*, *temp*, c_id, c_id[], c_id[][]

Examples:

```
compute          1 all centroid
thermo_style             step temp press c_1[0] c_1[1] c_1[2]
```

Description:

The compute results can be accessed by using c_id, c_id[], or c_id[][] for scalar, vector, or array, respectively.

2.2.24 timestep

Define the timestep.

Syntax:

```
timestep                 value
```

- value: timestep

Examples:

```
timestep                 0.01
```

Description:

Timestep can be redefined through the simulation.

2.2.25 unanalyze

Terminate an *analyze* command set before.

Syntax:

```
unanalyze                id
```

- id: analyze id

Examples:

```
analyze          1 Al ave/space 1 1 1000 z center 1 density/number file
   vol_Al.txt
unanalyze                1
```

Description:

2.2.26 uncompute

Terminate a *compute* command set before.

Syntax:

```
uncompute                id
```

- id: compute id

Examples:

```
compute          1 all temp
uncompute                1
```

Description:

2.2.27 undump

Terminate a *dump* command set before.

Syntax:

```
undump           id
```

- id: dump id

Examples:

```
dump         2 all atom 1000 dump.atom
undump       2
```

Description:

2.2.28 unfix

Terminate a *fix* command set before.

Syntax:

```
unfix            id
```

- id: fix id

Examples:

```
fix          1 all nve
unfix        1
```

Description:

2.2.29 units

Define the system units.

Syntax:

```
units            style
```

- style: *lj*

Examples:

```
units            lj
```

Description:

Currently, only *lj* units is available.

2.2.30 velocity

Define velocity to a group of particles.

Syntax:

```
velocity    group-id style args
```

- group-id: group id
- style: *create*, *set*

  create temp random-seed dist

  dist: *uniform*, *gaussian*

  set vx vy vz

Examples:

```
velocity     all create 1.0 72422 gaussian
velocity     freeze set 0.0 0.0 0.0
```

Description:

If the style *create* is used, it will create velocity profile to a group of particles.

If the style *set* is used, velocity of a group of particles will be set to exact numerical values.