Manual for running the DPD Codes

January 23, 2025

1 Instructions for running the codes stored in folder: t1_dpd_time_scaling

This section describe the scaling of time in dpd simulation, where you use the LAMMPS input file named `t1_time_scaling_dpd.in`.

Configuring system:

1. **Defining Variables:**

```
# Some variables
variable T    equal 1.0
variable rc   equal 1.0
variable rcD  equal 1.0
variable  L   equal 10
```

LAMMPS allows the definition of many types of variables. Some variables remain constant throughout the simulation and are defined globally, while others change dynamically at each time step whenever they are accessed to compute the motion of atoms. For every atom and at every time step, these dynamic variables are recalculated. However, the variables in this example (`T`, `rc`, `rcD`, `L`) remain constant throughout the simulation and for each particle.

2. **Essential Simulation Settings:**

```
# Basic simulation settings
units        lj
boundary     p p p
atom_style   atomic
dimension    3
```

These variables are essential and must be defined for every LAMMPS simulation. The command `units lj` sets all units such as mass, length, energy, temperature, and derived time in terms of the Lennard-Jones (LJ) potential specified. For example, if the LJ potential has a potential well depth of 3 units, then the temperature k_BT will also be expressed in these units. Thus, if the temperature is set to 4 units, it is 1 unit higher than the potential well depth defined in the simulation.

In DPD simulations, the LJ potential is not typically used. If the temperature is set to 1, then the temperature unit effectively equals the potential width unit. Consequently, all other variable values can be directly inferred, as they will also be implemented in terms of the temperature unit (k_BT).

The boundary condition `p p p` ensures periodic boundary conditions are applied. These conditions can also be switched off to allow for non-periodic boundaries. For example, an expanding boundary lets the simulation box grow as atoms diffuse away, while a non-periodic, non-expanding boundary results in atoms being lost once they cross the boundary. For more details on these options, refer to the LAMMPS manual.

The command `atom_style atomic` defines particles as point particles, meaning there are no considerations for electrons or nuclei. For example, `atom_style molecule` allows bonding between particles, but these bonds are classical (e.g., harmonic springs) and do not involve quantum mechanics. Similarly, `atom_style sphere` allows for angular momentum to be defined and applied, which is not possible with `atom_style atomic`. In future tutorials involving large Brownian particles, you will use `atom_style sphere` (or a hybrid style combining molecular bonding and angular momentum).

Finally, the `dimension` command allows you to specify either 2D or 3D simulations. In this case, and in most simulations, the dimension is set to `3`.

3. **DPD-Specific Variables:**

```
newton       on
comm_modify  vel yes
```

These commands are specifically required for DPD (Dissipative Particle Dynamics) simulations. When `newton` is set to `on`, pairwise interactions are stored and utilized by DPD during the simulation. As you have learned in your classes, DPD relies on pairwise forces involving surrounding particles. Even for computing the friction force, DPD identifies the nearest neighbors within the interaction range and uses their pair velocities to calculate the friction force.

By default, LAMMPS communicates the positions of neighboring particles. However, for DPD simulations, the velocities of neighboring particles are also required to calculate friction forces accurately. The command `comm_modify vel yes` ensures that the velocities of neighboring particles are communicated alongside their positions.

These settings are essential for the accurate implementation of DPD simulations, as they enable the proper calculation of dissipative and random forces based on both positional and velocity information.

4. **Creating a Simulation Box:**

```
# Create simulation box
region  your_Box block 0 ${L} 0 ${L} 0 ${L}
create_box  1 your_Box
```

Of course, to simulate particles, you will need something where the particles can exist and move a box that serves as the simulation domain. To efficiently handle particle interactions, LAMMPS maintains a list of nearest neighbors for each particle by dividing

the simulation box into grids. Each grid is allocated to a specific memory location to store particle IDs. However, this memory allocation is limited, which constrains the extent to which grids can expand. Therefore, you define a simulation box with specific dimensions. To avoid simulation artifacts caused by the finite size of the box, periodic boundary conditions (defined earlier) are commonly employed.

LAMMPS uses the `region` command to define simulation regions, which can take various shapes such as rectangular (using the `block` keyword), cylindrical (using the `cylinder` keyword), or others.

Once a region is defined, the `create_box` command is used to initialize the simulation box, which sets up the memory allocation for the particles. This command is crucial because it determines the degree of freedom and storage for the particles. For example:

- `create_box 1 your_Box` specifies a single type of particle system.
- `create_box 2 your_Box` would define a binary system with two types of particles.

If you need to include molecular systems with bonds, angles, or dihedrals, this memory allocation step ensures that each particle has the necessary space to store:

- The number of bonds per particle.
- The list of bonded neighbors.
- The dihedral bonds (if applicable), and more.

Note: If you encounter errors such as "list index out of range" or other memory-related issues, carefully review the `create_box` command in the LAMMPS manual. Often, such errors are due to insufficient memory allocation at this step.

By understanding and correctly implementing this command, you ensure that your simulation can handle the required complexity, whether for simple atomic systems or more complex molecular systems.

5. **Creating Atoms in a Defined Region:**

```
# Create_atoms 1 in region simBox
create_atoms 1 random 3000 12456 your_Box
```

The `create_atoms` command is used to generate atoms within a specified region. In the above example:

- `1`: Specifies the particle type.
- `random 3000`: Generates 3000 atoms randomly within the defined region.
- `12456`: Is the seed for the random number generator, ensuring reproducibility of atom positions.
- `your_Box`: Refers to the simulation box defined using the `region` and `create_box` commands.

It is important to ensure that all particles are generated within the boundaries of the simulation box. If you define another region larger than the simulation box and attempt to create atoms in that region, atoms will only be generated in the overlapping area between the box and the region. Any particles with randomly assigned positions outside the box boundaries will not be created.

This behavior prevents particles from existing outside the simulation domain, ensuring consistency and avoiding errors in subsequent simulations. To avoid such issues:

- Ensure the region used for `create_atoms` is entirely within the simulation box.
- Carefully set the dimensions of the box and region to match your simulation requirements.

Note: If you encounter missing atoms or an unexpectedly small number of generated particles, double-check the boundaries of the region used to generated the particle and the extent of the region used to generate the box. Misalignment between these can lead to partial or incomplete atom generation.

6. **Defining the Particle's Properties:**

```
mass         1 1.0
```

This command defines the mass of the particles in the simulation. In this example:

- `1`: Specifies the particle type.
- `1.0`: Assigns a mass of 1.0 (in simulation units) to the particles of type 1.

The mass plays a crucial role in determining the particle's dynamics, as it directly influences the acceleration, velocity, and position calculations through the equations of motion. Specifically, the relationship $F = ma$ connects the force (F) acting on a particle to its acceleration (a), using the mass (m).

Additional Properties:

- Other attributes, such as particle radius or rotational inertia, can also be defined if the chosen `atom_style` supports them.
- For example, if you want to simulate particles with a finite size or angular momentum, you can use the `sphere` atom style instead of `atomic`.
- The `atomic` atom style treats particles as point-like entities with no dimensions, so it does not allow for properties like radius or rotational inertia to be implemented.

If your simulation involves finite-sized particles or requires modeling their rotational dynamics, consider switching to an appropriate atom style, such as `sphere` or `hybrid`, depending on your needs. However, for simulations where particles are treated as point masses, `atomic` is sufficient and computationally efficient.

Note: When using different atom styles, make sure to refer to the LAMMPS documentation for the specific commands and parameters required to define additional properties like size, shape, or angular velocity.

7. **Defining the Interaction Among the Particles:**

```
# Define the pairwise interaction style (DPD)
pair_style       dpd ${T} ${rcD} 3854262
pair_coeff   1   1  78.0  4.5  1.0
```

This section defines the interaction between particles in the simulation. The `pair_style` command specifies the potential used for interactions between particles. In this case, the DPD (Dissipative Particle Dynamics) pair style is chosen:

- `dpd`: Specifies the DPD pair style, which is commonly used to simulate soft interactions between particles.
- `$T`: Represents the temperature parameter in the simulation.
- `$rcD`: Specifies the cutoff distance for the DPD potential.
- `3854262`: Is a random seed for generating the interaction coefficients, ensuring reproducibility of the simulation results.

The `pair_coeff` command specifies the interaction coefficients between particles. In this example:

- `1`: Refers to the first particle type.
- `1`: Refers to the second particle type.
- `78.0`: The strength of the interaction **A** or (a_{AA}).
- `4.5`: The friction coefficient, more details will come in the later sections.
- `1.0`: The cutoff range.

If you wish to disable inter-particle interactions, you can use the `pair_style none` command, which will treat the system as a "hot Lorentz gas," where the pressure and energy are defined by the ideal gas equation of state.

Common Pair Styles:

- `pair_style lj cut/off 2.5`: A widely used pair style with a cutoff distance of 2.5.

Each pair style has its own method for defining interaction coefficients, and once the `pair_style` is set, the `pair_coeff` command specifies the coefficients for the defined interaction. This tells LAMMPS how the particles of different types interact with each other.

Note: For a detailed description of these pair styles and coefficients, refer to the LAMMPS manual. The meaning and specific values of these parameters will be discussed in greater detail in the following sections.

8. **Defining the Integration Style, Simulation Time Step, and Initial Velocities:**

```
# Defining some simulation parameters
run_style    verlet
velocity all create ${T} 68768932
timestep 0.005
```

In molecular dynamics simulations, the equations of motion are integrated to compute the velocity and position of particles at each time step. However, a key challenge is determining the values of acceleration and velocity at the beginning (t) and end ($t + \Delta t$) of each time step. This is critical because small errors in force and velocity can lead to significant deviations in the system's behavior over long simulation times.

To resolve this, a formalism called the Verlet algorithm is used to accurately integrate the equations of motion. This is done using the `run_style verlet` command, which ensures the correct scheme for integrating the velocities and positions. If you do not specify the integration style, LAMMPS will default to its own integration method.

Initial Velocities: Before starting the simulation, you can initialize the velocities of the particles using a random distribution based on the Maxwell-Boltzmann distribution for a given temperature T. This helps the system reach equilibrium more quickly. The `velocity all create` T68768932 command generates random velocities for all particles, with the seed value 68768932 ensuring reproducibility of the random velocity distribution.

However, while this sets the system in motion with an equilibrium velocity distribution at the specified temperature, it does not guarantee that the system is in thermodynamic equilibrium right away. The position-dependent potential energy, defined by the pair interaction style, will require time to equilibrate. During this process, the system will adjust until the energy is evenly distributed across all degrees of freedom, with each degree storing energy $\frac{k_B T}{2}$ according to the equipartition theorem.

Time Step Selection: The time step (Δt) plays a crucial role in the accuracy and efficiency of the simulation. A small time step ensures accurate integration but increases the simulation time to reach the same physical duration. Conversely, a larger time step can cause issues, such as overlapping particle interactions and infinite acceleration, leading to particles moving outside the simulation box and resulting in errors.

Therefore, it is essential to choose a time step that strikes a balance between accuracy and computational efficiency. The time step should be small enough to accurately capture the dynamics of the system, but not so small that it unnecessarily increases the computational load. As a general guideline, use a time step that is sufficiently small to ensure stability but large enough to avoid excessive computational costs.

9. **Showing the Parameters on Terminal:**

```
# Output thermodynamic quantities which shows on terminal
thermo_style custom step time temp press
thermo       100
```

The `thermo_style` command in LAMMPS allows you to customize the thermodynamic quantities displayed during the simulation. By default, LAMMPS calculates various quantities, such as time, temperature, pressure, and kinetic energy, without the need for any additional `compute` commands. These quantities are automatically displayed in the terminal unless you specify otherwise.

In this case, the command `thermo_style custom step time temp press` specifies that the following quantities will be shown:

- `step`: The current simulation step.
- `time`: The current simulation time.
- `temp`: The temperature of the system.
- `press`: The pressure of the system.

The command `thermo 100` indicates that LAMMPS will print these values every 100 simulation time steps. This is useful for monitoring the progress of the simulation and checking for convergence or any unusual behavior in the system over time.

By adjusting the `thermo` value, you can control the frequency of the thermodynamic output. For example, setting `thermo 1000` would print the specified quantities every 1000 time steps, while a smaller number would result in more frequent output.

10. **Apply an Integrator:**

```
# Apply the NVE ensemble
fix your_fix all nve
```

In LAMMPS, integrators are applied using `fix` commands. A `fix` can be applied to a selected group of atoms, and if no specific group is specified, it will be applied to all atoms in the system. In this example, the `fix your_fix all nve` command applies the NVE ensemble to all atoms. The `all` keyword indicates that the fix is applied to all atoms in the system.

Fixes are essential for updating the particle positions and velocities during the simulation. However, not all fixes are integrators. The integrators that are used to move particles and update their properties are specific commands, such as:

- `nve`: For constant volume and energy.
- `nvt`: For constant volume and temperature (Langevin dynamics).
- `npt`: For constant pressure and temperature.
- `rigid/small molecule`: For simulating rigid molecules.

Without an integrator, the system will not evolve, and the simulation will not update particle positions or velocities, resulting in a stationary system.

You can also remove a previously applied `fix` by using the `unfix` command. This allows you to apply different fixes sequentially, such as compressing a box using the `Npt` fix and then switching to the `NVT` ensemble after unfixing the previous fix.

Note: Make sure to choose the appropriate integrator based on the type of simulation you are running and the physical conditions you want to maintain (e.g., constant temperature, constant temperature and pressure, etc.).

11. **Run the Simulation:**

```
# Equilibration phase - run to allow the system to stabilize
run          200000
```

Once you have set up your simulation with the appropriate parameters and applied the necessary fixes, you can begin running the simulation by specifying the number of steps. In this case, the command `run 200000` will execute the simulation for 200,000 steps.

The simulation time with each simulation step advances in multiples of the time step (dt) defined earlier with the `timestep` command. For example, if $dt = 0.005$, the simulation will progress by 0.005 units of time with each step.

Typically, the first part of the simulation is used for equilibration, where the system stabilizes, and the particles reach an equilibrium state. After equilibration, you may perform additional runs to collect data or to explore specific properties of the system.

This is when you would apply the commands to save data or modify the simulation conditions for further analysis.

Note: Be sure to check that the system has equilibrated properly by observing relevant thermodynamic quantities (e.g., temperature, pressure) and ensuring that they remain stable over time before proceeding with data collection or further analysis.

12. **Using Built-In Computational Facilities of LAMMPS:**

```
# Compute MSD (Mean Squared Displacement) after equilibration
reset_timestep  0
compute your_msd_all all msd
fix your_msd_output all ave/time 100 1 100 c_your_msd_all...
...file your_msd.dat mode vector
```

After running the equilibration phase, you may choose to reset the timestep to 0 using the command `reset_timestep 0`. This is useful as it makes further computations easier and ensures that the time tracking begins from a fresh point.

LAMMPS provides a variety of built-in `compute` commands that allow you to calculate various physical quantities. For instance, the `compute msd` command computes the Mean Squared Displacement (MSD) of the particles, which is a common measurement in molecular dynamics simulations to study particle diffusion. You can find more information about different compute commands in the LAMMPS manual.

Once you define a compute variable, you need to apply an appropriate averaging and output mechanism to record the data. The `fix ave/time` command is used for this purpose, which averages the computed quantity over time and writes it to a file. In the example provided, the command:

```
fix your_msd_output all ave/time 100 1 ...

...  100 c_your_msd_all file your_msd.dat mode vector
```

does the following:

- Averages the MSD (computed by `compute your_msd_all all msd`) over 100 steps, with output every 1 step.
- Writes the result to a file named `your_msd.dat` in a vector format.

Note: The `c_your_msd_all` in the command refers to the output of the `compute` command and is called using a specific format (e.g., `c_` prefix). Different types of variables (e.g., scalar, vector) are called in distinct ways, which you can learn from the LAMMPS manual.

For more details on how to use the `compute` and `fix ave/time` commands, refer to the LAMMPS documentation.

13. **Running Again to Collect the Data After Ensuring System Equilibrium:**

```
# Production run to measure MSD over time
run          200000
```

After ensuring that the system has reached equilibrium, you can begin the production run to collect data. In this phase, you will measure the desired quantities, such as MSD (Mean Squared Displacement), over time.

Once all the necessary quantities are defined (e.g., using `fix` commands) and the system has equilibrated, you can execute the production run. The command `run 200000` will run the simulation for 200,000 time steps, allowing you to gather the data you need.

During the production run, the system is typically monitored for further stability, and data is output as specified by the commands in your input script (e.g., using `fix ave/time` to average quantities like MSD).

Steps to Follow:

To perform the scaling of time, students can follow the following steps:

1. Run the code stored in the folder named `t1_time_scaling_dpd.in`.

2. Run the Python code `t1_msd_data_rectifier_and_plotter.py`. The code will analyze the output MSD data and print the diffusivity. It will also export a plot with relevant information.

3. You will have MSD in DPD length units for the DPD beads. Use the relation to convert the MSD into real units:

$$t_{\text{phy}} = \frac{N_m \cdot (30.0\rho N_m)^{2/3} \langle R^2_{\text{cm}} \rangle}{D_{\text{water}}} \quad \left(\text{in units of } \hat{A}^2/\text{s}\right)$$

   Where R^2_{cm} is your MSD for the DPD beads obtained from your Python code. The factor N_m adjusts the center-of-mass bead displacement to the displacement of particles constituted by the beads. The factor $\cdot(30.0\rho N_m)^{2/3}$ converts the total displacement of the DPD particles (which is $N_m R^2_{\text{cm}}$) to real time units in $\hat{A}$. D_{water} is the diffusivity of water given in real units for actual water molecules.

   Length is scaled by the volume of ρN_m (the number of DPD molecules in one unit volume, in DPD length). Here, $30\hat{A}^3$ is the volume of a single molecule in the real system. So, one unit volume in DPD corresponds to $30 \cdot \rho N_m \hat{A}^3$ in real units. Consequently, one length unit in DPD is equal to $(30 \cdot \rho N_m)^{1/3}\hat{A}$.

4. Once you compute the physical time in real units covered during the given simulation time, you can determine the relation between the simulation physical time and the real physical time using the expression:

$$t_{\text{f}} = \frac{t_{\text{phy}}}{t_{\text{sim}}}$$

5. Once you have t_{f}, you can convert any simulation time to the real time units by multiplying t_{sim} by the factor t_{f}.

   Note: Students must not confuse the number of simulation steps with the simulation time t_{sim}. The simulation time t_{sim} is given by $t_{\text{sim}} = \text{simulation\_steps} \cdot dt$.

2 Instructions for Running the Codes Stored in Folder: e1_rho_vs_pressure

Since you can define your time scale for any force constant A and density ρ, so that you get the same diffusivity from DPD, after unit conversion, and equal to the diffusivity value in the real system, what will be the appropriate choice of force constant and densities? The related questions asked in your exercise exactly point to this aspect.

By solving this exercise, you will come to know about the appropriate ρ value.

Once you have ρ fixed, you can fix the force constant A (or a_{AB}), by using the relation between κ^{-1} and A, for a given value of ρ, α (which you will also calculate in this exercise).

So in practice, you first fix density ρ and then force constant a_{ij} with an appropriate value of σ (the strength of noise). While the cutoff is usually set to 1 for simplicity. Then calibrate your length and time scales accordingly, from the diffusion coefficient calculated from the simulation, as you did in your tutorial.

But first, please read the below explanation of the system configuration, which focuses more on the pair coefficient defined in your LAMMPS script file `e1_rho_and_pressure.in`.

Configuring the System:

To simulate DPD in your LAMMPS package, you use the following commands:

1. **Defining DPD Pair Coefficient in LAMMPS:**

```
# Define the pairwise interaction style (DPD)
pair_style       dpd ${T} ${rcD} 3854262
pair_coeff   1   1  25.0  4.5  1.0
```

It is easy to guess how LAMMPS implements values in the code if you keep T as 1. As in the LJ units, energy and temperature are defined in ϵ units, where ϵ is the potential depth of the LJ potential. If you define k_BT (which we call T) as 1, then you can guess what the other quantities like force constants will be. For the full details, please see the LAMMPS manual. Always keep **`rcD`** greater than the cutoff defined in the **`pair_coeff`** section.

You have defined **`pair_coeff`** between particle type 1 and 1 (a_{AB}, where $A = 1$; type I particle, $B = 1$; also type I particle), which is the only particle type considered in your simulation. If you have more particle types, you will need to define coefficients for all the combinations of the particle interaction types; otherwise, LAMMPS will show an error.

In your code, you have defined the conservative force constant a_{AB} (also represented as A) as 25.0.

To ensure proper thermo-stating, you will need to define a good σ_{dpd} (please do not confuse with the representation of length often by the same symbol) coefficient. In LAMMPS, you define γ, which in your example script is defined as 4.5. The σ is related to γ by the expression $\sigma = \sqrt{2k_BT\gamma}$. So for a defined $\gamma = 4.5$, you have σ implemented as $\sigma = 3$. For larger force coefficients A (or a_{AB}), more caging will occur and the system will take more time to equilibrate. Therefore, you will need to choose a higher value of σ. For

example, for the force constant $A = 25$, an appropriate value of σ is 3. In other words, in LAMMPS, you define the corresponding $\gamma = 4.5$ as shown in the example code.

The cutoff length in your example code is defined as 1.0. It is a usual practice to make length scale calibration easier. It should always be less than `rcD`.

Steps to follow:

Please follow the steps below to extract the ρ vs. p (pressure) data. The mathematical relation between these two quantities is often referred to as the EOS (Equation of State). Using the same data, you can solve the related problems in your exercise.

1. Run the script file `e1_rho_and_pressure.in`. All students are advised to select a unique random force constant A (or a_{AB}) within the range $25 - 75$. Any form of copying will result in a reduction of marks.

2. To set different values of ρ, students can either change the box length in the script, as specified by the line `variable L equal 10`, or change the number of particles created in the box by modifying the command line `create_atoms 1 random 3000 12456 your_simBox`. Since $\rho = N/L^3$ (where 3000 is the number of particles), changing the box size is the recommended approach, as increasing the number of particles will also increase the simulation time.

3. For each value of ρ, students will need to run the simulation multiple times and observe the terminal output where the pressure is printed. The pressure should converge to a certain value (with some fluctuations, which are acceptable). After sufficient simulation time, take any stable value of the pressure, and this will be the pressure corresponding to the chosen ρ value that you implemented by adjusting the simulation box or the number of particles.

4. Once you obtain a set of ρ and p values by running the code for different values of ρ and a specific force constant A, you should record these values in two columns in the file `e1_rho_vs_pressure.txt`. Each row should contain a set of ρ and the corresponding pressure value.

5. After creating the text file, students should open the Python script `e1_plotter.py`. In the line `a = 25` within the script, replace the force constant value with the value you used in your LAMMPS script. Then, run the Python code to generate the plot. From the plot, you can calculate the value of α, which is a constant given by the expression $\alpha = p - \frac{\rho}{A\rho^2}$.

3 Instructions for running the codes stored in folder: e1_binary_mixture

In this section, you will establish the relationship between the simulation parameters and the theoretical quantities that govern the evolution of the polymer system in the binary mixture. This involves customizing and reconfiguring your LAMMPS code `e1_flory_parameter.in`.

System configuration:

To create a binary system of polymers, you will introduce another type of bead into the system and define the interactions among the different bead types, as explained in detail below.

1. **Creating memory space for the two types of particles:**

```
### Define simulation box
variable     Lx equal 20
variable     Ly equal 10
variable     Lz equal 10

region       your_box_total block 0 ${Lx} 0 ${Ly} 0 ${Lz}
create_box   2 your_box_total  # Binary system with 2 particle types
```

   In this relevant section of your LAMMPS script `e1_flory_parameter.in`, you are defining a rectangular box that is extended more in the x-direction. The command `create_box 2 your_box_total` allocates memory for two types of particles or beads.

2. **Creating two types of particles in the designated regions:**

```
# Regions for particle distribution
region       region_A block 0 10 0 ${Ly} 0 ${Lz}
region       region_B block 10 20 0 ${Ly} 0 ${Lz}

# Create particles in each region
create_atoms 1 random 3000 12345 region_A  # Type A: atoms in region A
create_atoms 2 random 3000 67890 region_B  # Type B: atoms in region B
```

   Now you define two sections in the box along the x-axis: one from the length $0 - 10$ and another from $10 - 20$. You create atom type 1 in region A and atom type 2 in region B. However, keep in mind that particles are not restricted to live in any particular region; they are free to move throughout the entire simulation box defined by the `create_box` command.

   We make this distinction because if the system phase separates, it will do so randomly within the simulation box. Tracking the regions and counting the number of particles of a specific species in a particular region would be difficult. However, if the system is already separated, then particles in each phase will not mix. As a result, it will be easier to count the density of each species in the separated regions, since these regions often remain static during the simulation.

Note: If you define a region that extends beyond the area used in the `create_box` command, particles generated within the specified regions that exceed the box boundaries will be excluded. Consequently, fewer particles will be generated. LAMMPS will not throw an error in this case, and the code will run with the reduced number of particles. Be mindful of this when redefining regions, as the newly defined region must be a subset of the box region used by the `create_box` command.

3. **Defining the pair coefficient among the different types of particles:**

```
### Define DPD pair style
pair_style   dpd ${T} ${rcD} 3854262
pair_coeff   1 1 25.0 4.5 1.0  # A-A interactions
pair_coeff   1 2 45.0 4.5 1.0  # A-B interactions (adjust as needed)
pair_coeff   2 2 25.0 4.5 1.0  # B-B interactions
```

Now, we reach the most crucial part, which you will be modified during your exercise. In the script, you can see that the pair coefficients for all combinations of the two types of particles are defined using their particle type IDs. Note that the particle type ID is not the same as the particle ID in LAMMPS. A particle type ID represents all particles of the same type in the system.

The coefficient A or a_{AB}, which represents the strength of the repulsive force, will influence the mixing and de-mixing behavior of the polymers in your system. If we set the values $A_{11} = A_{22}$ (which is sat as 25.0 in your script), then, using the mean-field expression, we find that the Flory parameter χ is linearly related to the quantity $A_{12} - A_{11}$. To induce more de-mixing in the system, you will increase the value of A_{12} (which will ultimately correspond to increasing the χ value). This is reflected in the command `pair_coeff 1 2 45.0 4.5 1.0`, in the example script, where $A_{12} = 45$, which will be the applied value of a_{AB} in your simulation script as asked in the exercise question.

Steps to follow:

To solve the relevant questions, students are advised to follow the steps below:

1. Run the LAMMPS script `e1_flory_parameter.in` after setting the desired value for the A_{ij} pair coefficient.
2. You can visualize the system using OVITO. Simply load the data file into OVITO and adjust the particle size, color, etc. You will be taught this in your class, so please make sure to note down the steps during the class.
3. The script will export all the data into an output file named `simulation_data.lammpstrj`.
4. Within the same folder (`e1_density_distribution.py`), you will find a Python script that will give you the value of the density and x, the fraction of the density of type B particles in a phase-separated region.
5. Fill the `x_vs_delta_a.txt` file, where first column must be x and the second column must be Δa. It will calculate Δa vs χ and give you the desired plot.
6. Use the fraction of the density (x) to calculate the Flory parameter χ using the expression:

$$\chi = \frac{\ln\left[\frac{(1-x)}{x}\right]}{1-2x}.$$

7. For different values of $A_{12} - A_{11}$ ($= a_{\mathrm{AB}} - a_{\mathrm{AA}}$), where $A_{11} = A_{22} = 25$ is always kept constant, calculate the value of χ. You will then have a list of χ vs $\Delta a = A_{12} - A_{11}$ data, which you can plot to determine the value of β in the given expression $\chi = \beta \Delta a$.

Appendix

Using OVITO for Visualization

1. Load the trajectory file: `File > Load File`.
2. Apply `Modifiers` to enhance visualization (e.g., `Color Coding`).
3. Adjust the perspective for better screenshots.

Using `matplotlib` for Plotting

1. Install `matplotlib` using `pip install matplotlib` if not already installed.
2. Run the provided plotting script, replacing placeholders with actual data.
3. Save the generated plot as a PNG file.
