UNIVERSITY OF HELSINKI
FACULTY OF SCIENCE

Course: Programming Projects in Molecular Modeling

Project 3

Author: Mahdi Torabi, Md. Shafayet Islam

Instructor: **Prof. Alex Bunker**

Date: March 25, 2025

Contents

1 Introduction

In this project, we have investigated the dynamics of particles in a box with periodic boundary conditions (PBC), interacting with each other via Lennard-Jones potential (LJ). In addition, we implemented the steepest descent algorithm with an adaptive time step that modifies the time step based on the maximum force to minimize the system beginning from the last frame of the trajectory and stopping when the maximum number of steps is reached or force is less than an epsilon value or the difference between the energy of the current step and the previous one is less than an epsilon.

Next, the linked cell algorithm was employed to increase the efficiency of the code and accelerate the simulation for larger systems as this algorithm scales linearly with the number of particles in the system. We also employed some programming tricks to make the code run faster. In both the 2D and 3D boxes, the scaling of the computation time with the box size was benchmarked for both the naive approach and the linked cell algorithm and the difference in scaling between the two was shown. The step size in all simulations was taken to be 0.0001 as it was required in the project description.

All the code is written in `C++` programming language with a *make* file provided for easy compiling of the source code. All simulations are seeded to ensure the reproducibility of the results. The description of the program structure is in section A. All the coordinates are in *xyz* file format and the energy data are in a simple text format with *.dat* as a file extension to ensure compatibility with *xmgrace*. All the visualizations were performed by *VMD* application and the plots were made using a simple `Python` script.

2 Part 1 and 2

2.1 Theory

2.1.1 Lennard-Jones Potential

The Lennard-Jones (LJ) potential is a function used in physics and chemistry to describe the interaction between a pair of neutral atoms or molecules. It models van der Waals forces (attractive forces at long distances) and Pauli repulsion (repulsive forces at short distances) in molecular dynamics simulations. The LJ potential mathematical form is:

$$U(r) = 4\epsilon \left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right] \tag{1}$$

where ϵ is the potential well's depth, representing the interaction's strength. σ is the finite distance at which the interparticle potential is zero (it roughly corresponds to the diameter of the particles). The term $\left(\frac{\sigma}{r}\right)^{12}$ represents the short-range repulsion due to overlapping electron clouds (Pauli exclusion principle). The term $\left(\frac{\sigma}{r}\right)^{6}$ represents the long-range attraction due to van der Waals forces.

The force is calculated as the negative gradient of this potential.

2.1.2 Steepest Descend algorithm

Steepest descent is a first-order iterative optimization algorithm that uses the potential energy gradient to update the position at each step. It is a common algorithm to perform energy minimization in various simulations and machine learning. The motion is in the direction of the negative gradient of the energy function as the gradient points toward the maximum increase in the function.

Given LJ potential $U(\mathbf{r})$, the goal is to find the minimum of U by iteratively updating $\mathbf{r}$ using the steepest descent method:

$$\mathbf{r}_{k+1} = \mathbf{r}_k - \nabla U(\mathbf{r}_k)\Delta t \tag{2}$$

where $\nabla U(\mathbf{r}_k)$ is the gradient of the potential energy function.

If we consider the algorithm as above, convergence will be slow; therefore, adjustment of the step size based on the maximum force will accelerate this process. The position update will be then as follows:

$$\mathbf{r}_{k+1} = \mathbf{r}_k - \frac{\nabla U(\mathbf{r}_k)}{\max(|\nabla U(\mathbf{r}_k)|)}\Delta t \tag{3}$$

2.1.3 Velocity Verlet algorithm

For the integration of the equation of motion, we used the velocity verlet algorithm. The algorithm is as follows:
The position is updated:

$$r(t+\Delta t) = r(t) + v(t)\Delta t + \frac{1}{2}a(t)(\Delta t)^2 \tag{4}$$

$$v\left(t+\frac{\Delta t}{2}\right) = v(t) + \frac{1}{2}a(t)\Delta t \tag{5}$$

$$a(t+\Delta t) \quad \text{(computed from forces)} \tag{6}$$

$$v(t+\Delta t) = v\left(t+\frac{\Delta t}{2}\right) + \frac{1}{2}a(t+\Delta t)\Delta t \tag{7}$$

The velocity update can also be written as follows:

$$v(t+\Delta t) = v(t) + \frac{1}{2}\left[a(t) + a(t+\Delta t)\right]\Delta t \tag{8}$$

The benefit of this algorithm is that It has an error of $O(\Delta t^3)$ in energy but $O(\Delta t^2)$ in position.

2.2 Method, Results and Discussion

2.2.1 Two Dimensions

In this part, many different setups were used. Let's go through them one by one. First Let's take 20 Particles:

A 20 particle simulation was performed with a box size of 10 by 10, with initial position as a square lattice and initial velocity in the range $[-0.6, 0.6]$. Any higher initial velocity led to system destabilization and consequently the failure of the simulation. The simulation run was 10^6 steps with a step size of 0.0001. A minimization is then performed on the last frame of the trajectory with PBC removed and the steepest descent algorithm is used. Maximum steps of 10^7, a tolerance for the force to be 10^{-4}, and a tolerance of 10^{-7} for energy difference between two consecutive minimization steps were used. If any of these conditions are met then the minimization will stop and the final result will be written in an *xyz* file format.

The total energy plot (kinetic + potential) of the simulation is in fig. 1 and for the minimization in fig. 2. The minimized configuration is in fig. 3.

Taking a look at fig. 1, we see that energy drops at the beginning of the simulation which is expected as the system relaxes and the particles move toward a lesser energy configuration. During the simulation, the energy oscillates around an average value, and the average value of a particle is more or less constant throughout the simulation.

The minimization process is also performed properly as shown in fig. 2 where the energy of the system reaches a minimum before the end of the minimization to make sure that the minimization is not stopping prematurely. The minimum configuration is in fig. 3 which shows that the spherical particles prefer to pack in a triangular lattice in a 2D space. This ensures the minimum energy of the system as in such configuration, the particles are equidistant from each other, ensuring the most efficient packing and providing the highest packing density for equal-sized particles.

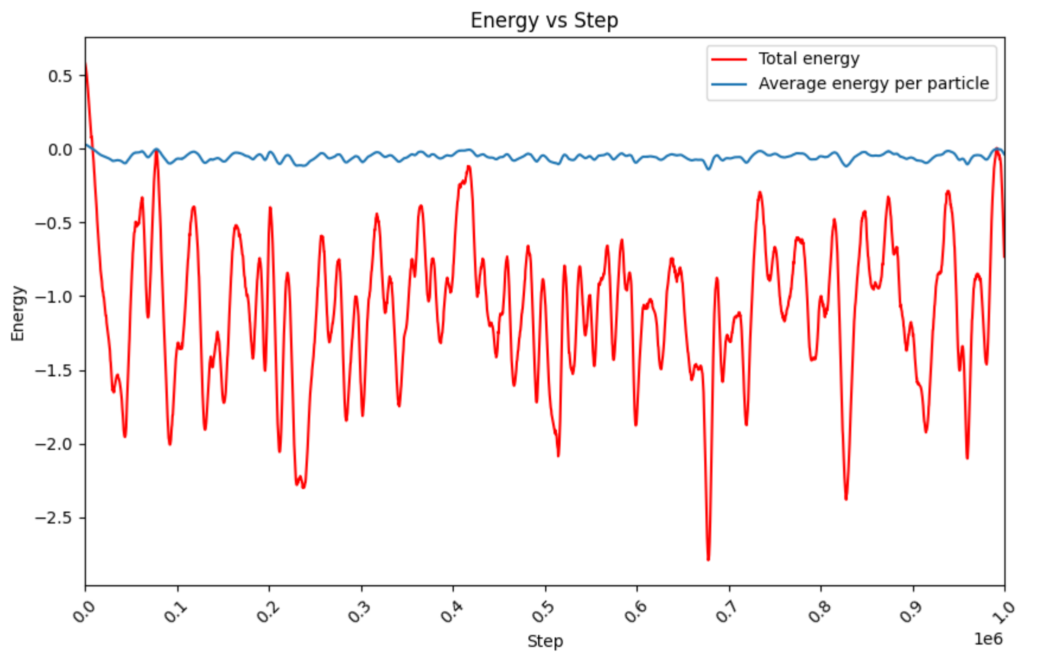

Figure 1: The total energy of the system and the average energy per particle along the simulation for 20 particles in a 2D box.

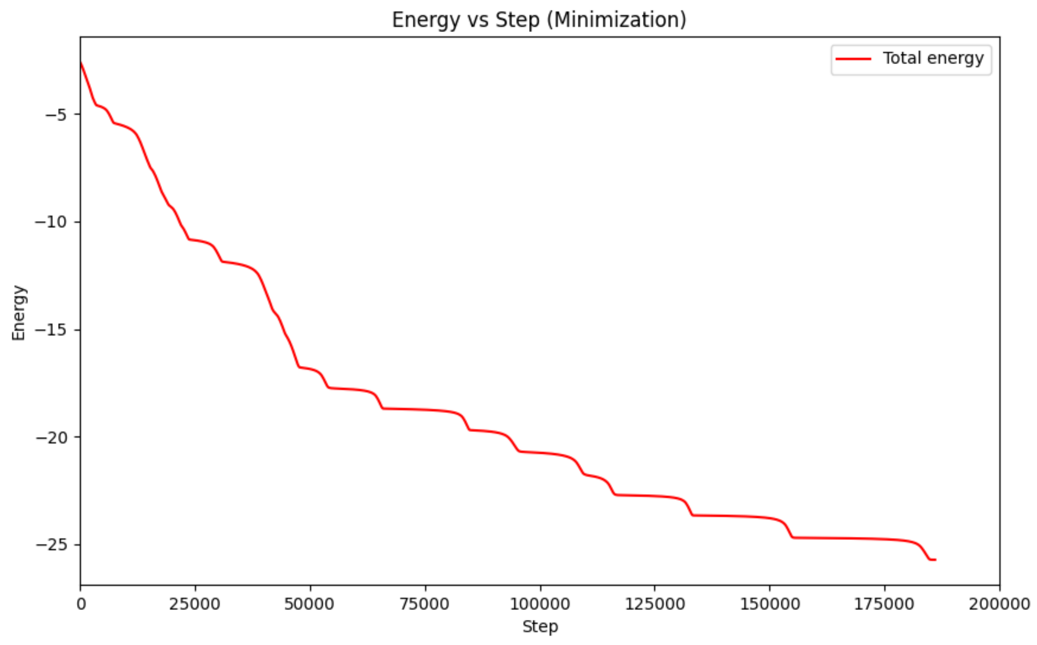

Figure 2: Energy of the system with 20 particles and no PBC conditions during the minimization procedure.

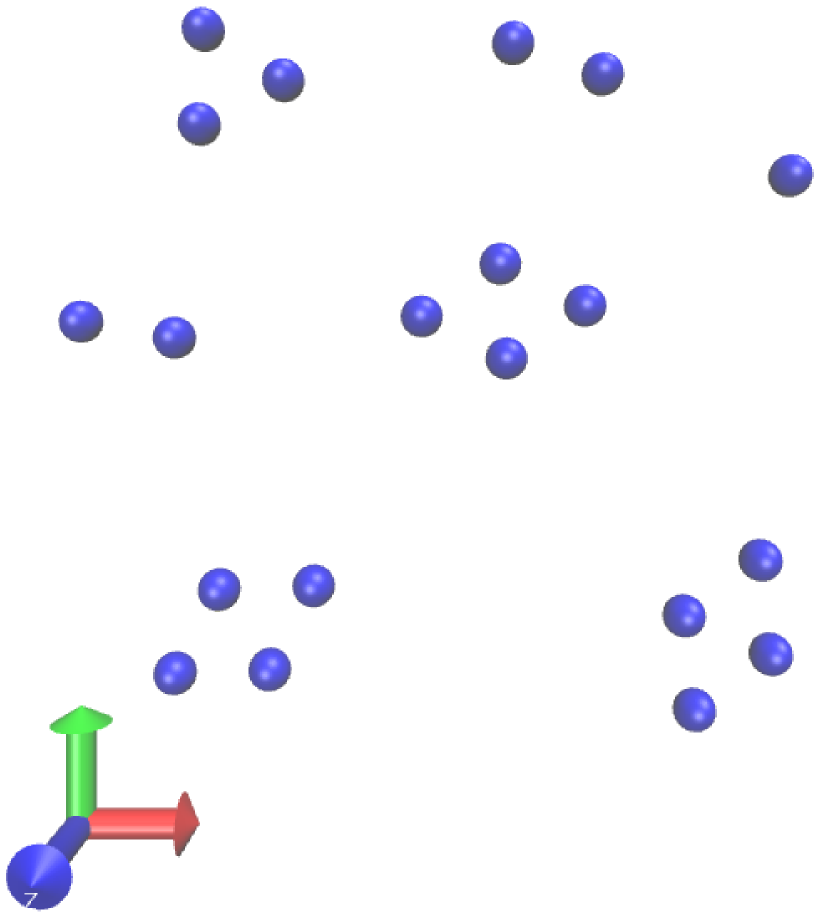

Figure 3: The final configuration of the system with 20 particles in 2D after minimization.

Now, Let's take 30 particles in the same box size, increasing the density to 0.3. The same procedure was applied except for this number the initial velocity had to be in the range $[-0.4, 0.4]$ otherwise the simulation would fail. The energy plot is in fig. 4 and for the minimization process in fig. 5. The configuration with minimum energy is in fig. 6. We can see that the energy of the system is lower than the 20 particle case. The energy per particle is more stable. The minimization was successful and a minimum was reached. The minimized configuration is the same as previously with a triangular (or hexagonal) lattice formation that is more pronounced.

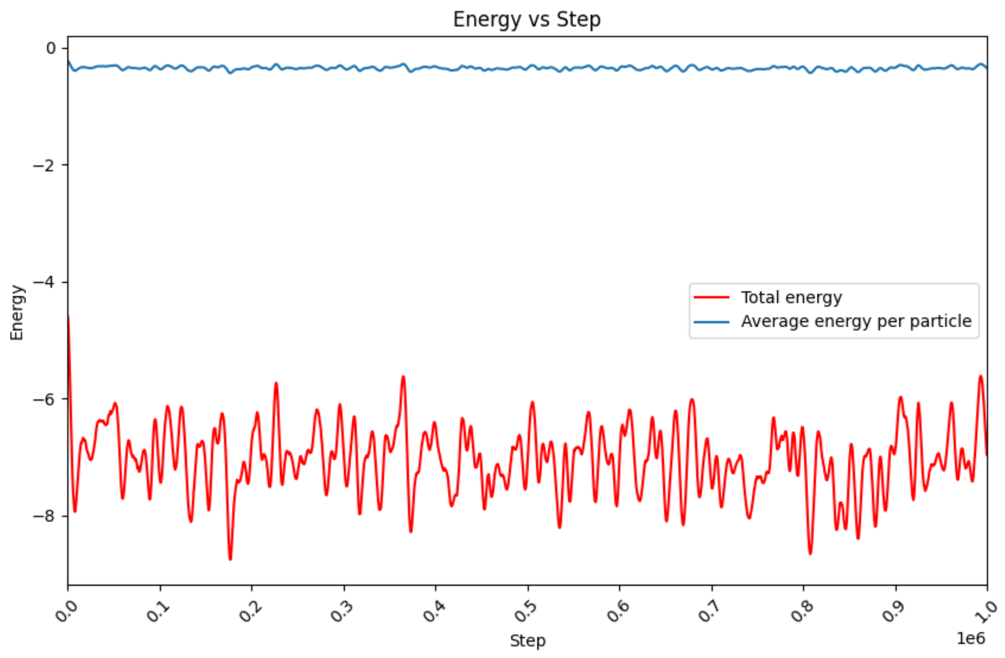

Figure 4: The total energy of the system and the average energy per particle along the simulation for 30 particles in a 2D box.

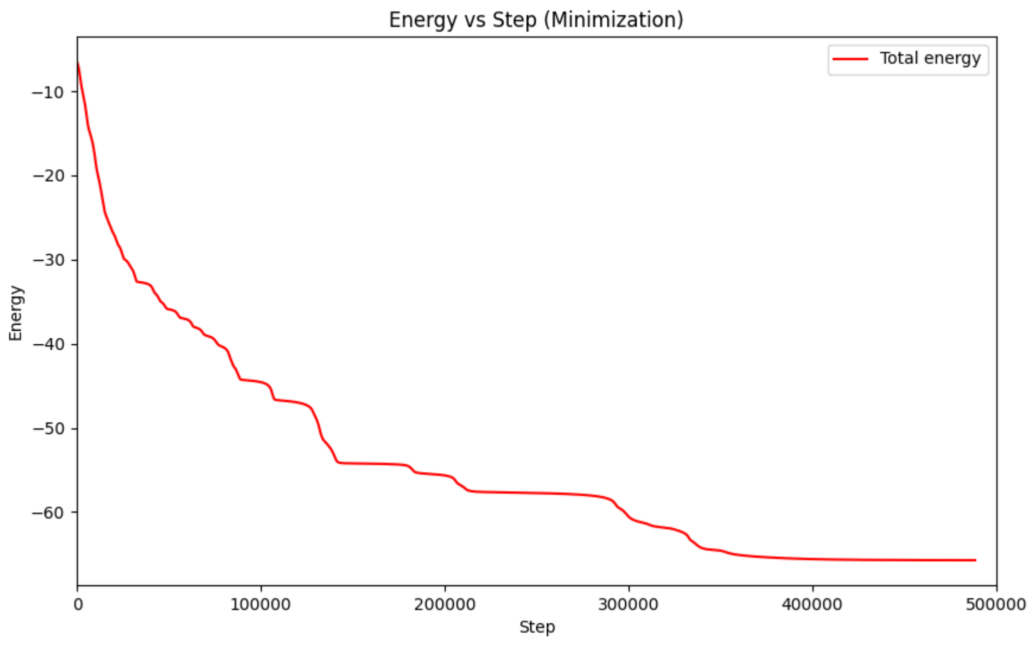

Figure 5: Energy of the system with 30 particles and no PBC conditions during the minimization procedure.

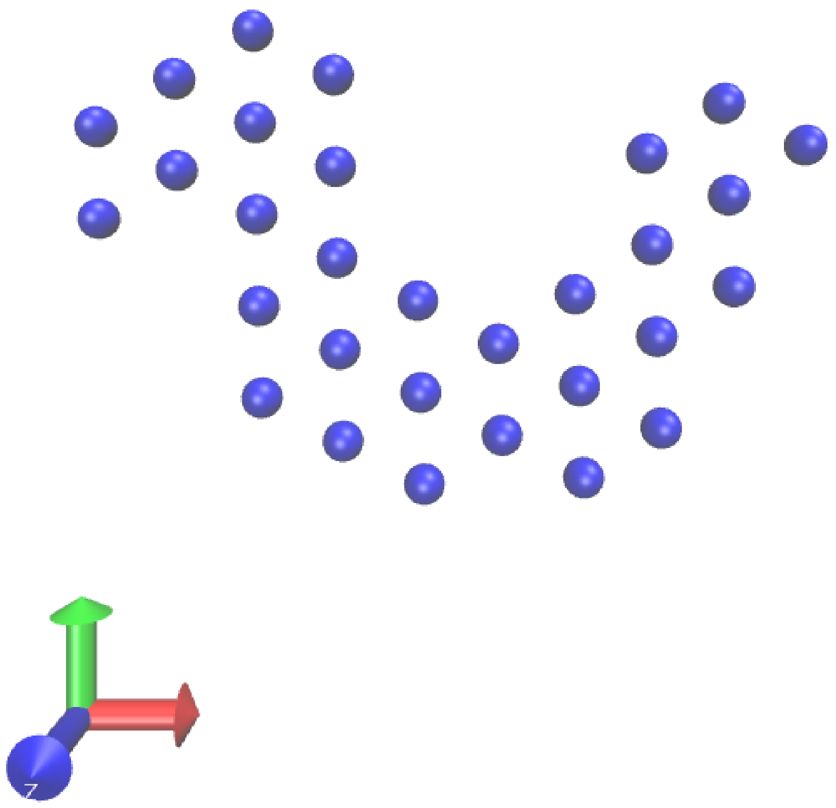

Figure 6: The final configuration of the system with 30 particles in 2D after minimization.

Finally, Let's take 40 particles in the same box size (density is 0.4 now):
Despite many tries to make this simulation succeed, by limiting the initial velocity range, this simulation was only stable for a short time. A solution to overcome this difficulty is to initialize the position in a triangular lattice as opposed to a square lattice. This was inspired by the minimized configuration of the previous runs. In this way, all the particles are equidistant from each other as opposed to the square lattice as the particles on the diagonal are further from each other than the two particles that lie on the side.

However, even with this method and reducing the initial random velocity range to $[-0.1, 0.1]$, the simulation is stable within the 10^6 steps that were for all the previous simulations. This is enough for energy analysis and visualization. However, the system will not be stable for a very long time as it will also collapse around $2 \cdot 10^6$ steps.

The energy plot is in fig. 7 and for the minimization process in fig. 8. The configuration with minimum energy is in fig. 9. we can see that the fluctuation in energy levels is less than in the previous attempts, and the minimized system shows the efficient packing of the triangular (or hexagonal) packing in a 2D box.

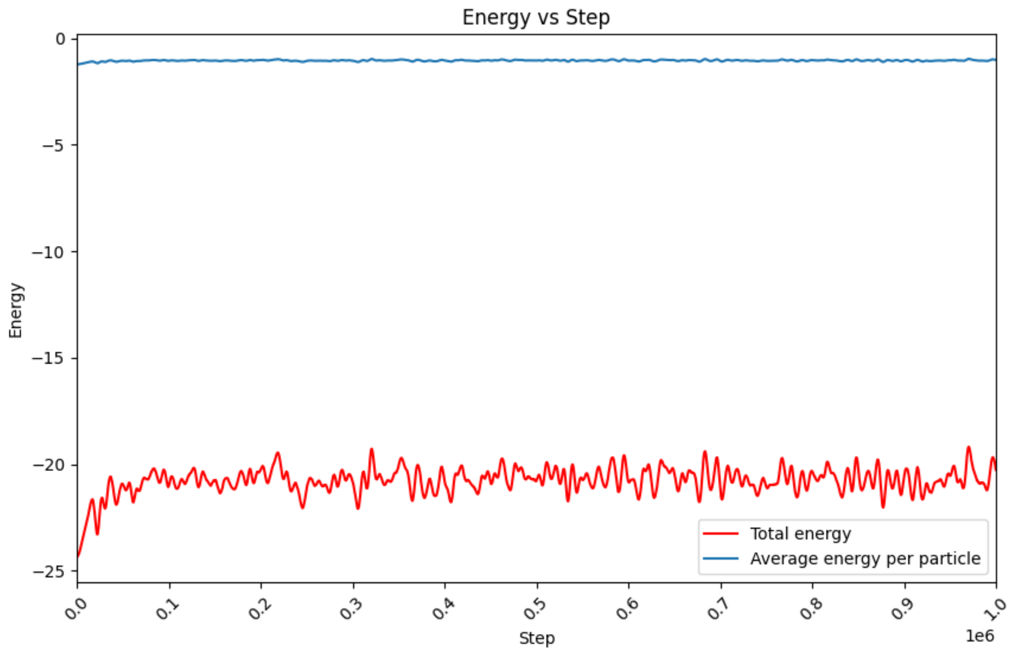

Figure 7: The total energy of the system and the average per particle along the simulation for 40 particles in a 2D box.

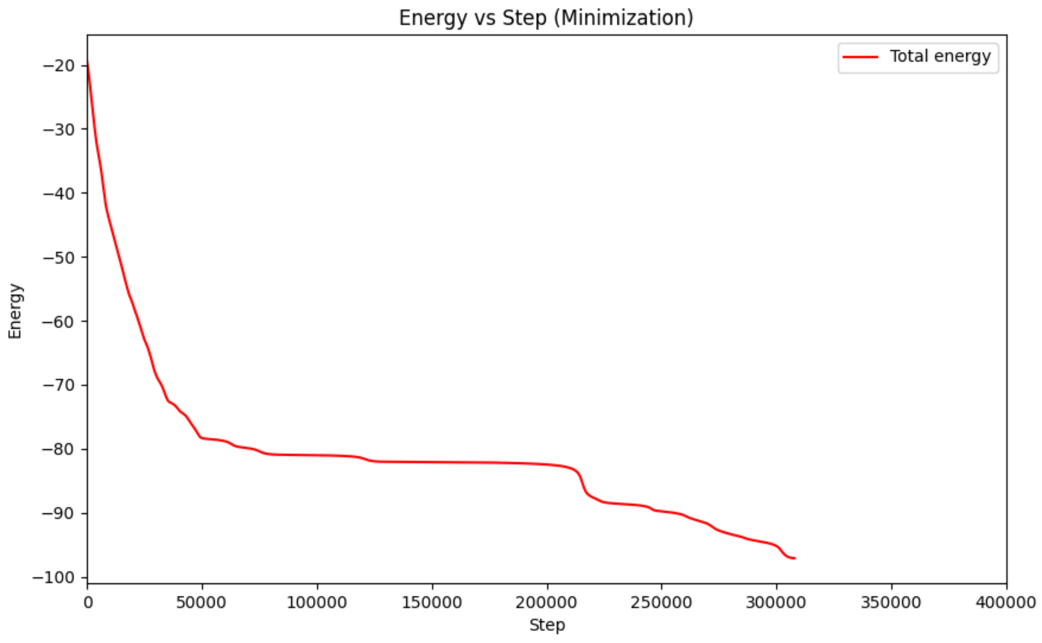

Figure 8: Energy of the system with 40 particles and no PBC conditions during the minimization procedure.

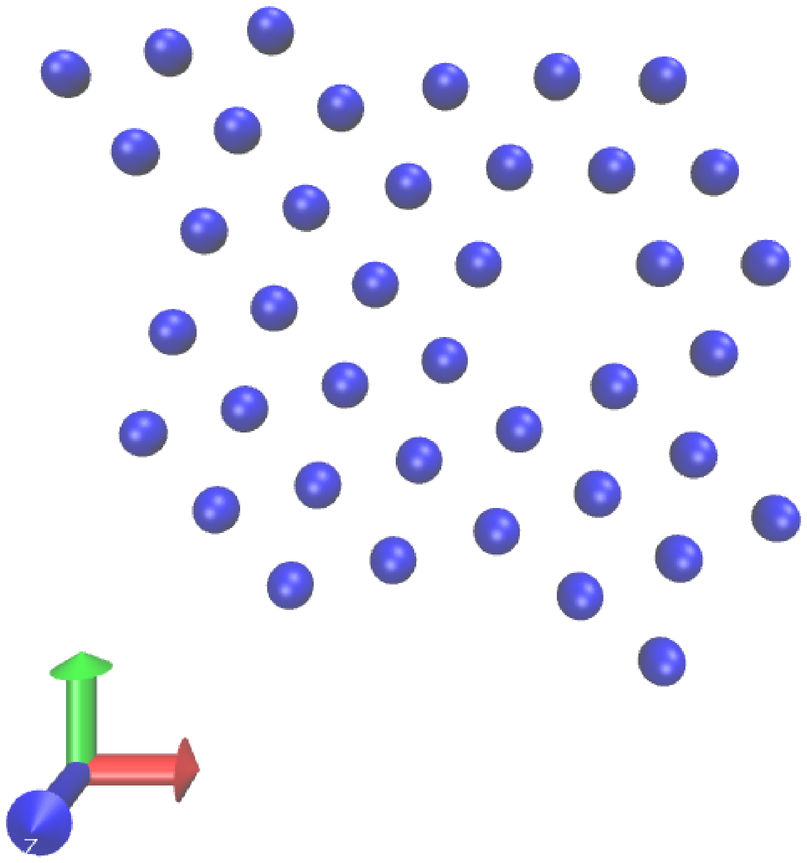

Figure 9: The configuration of the system with 40 particles in 2D after minimization.

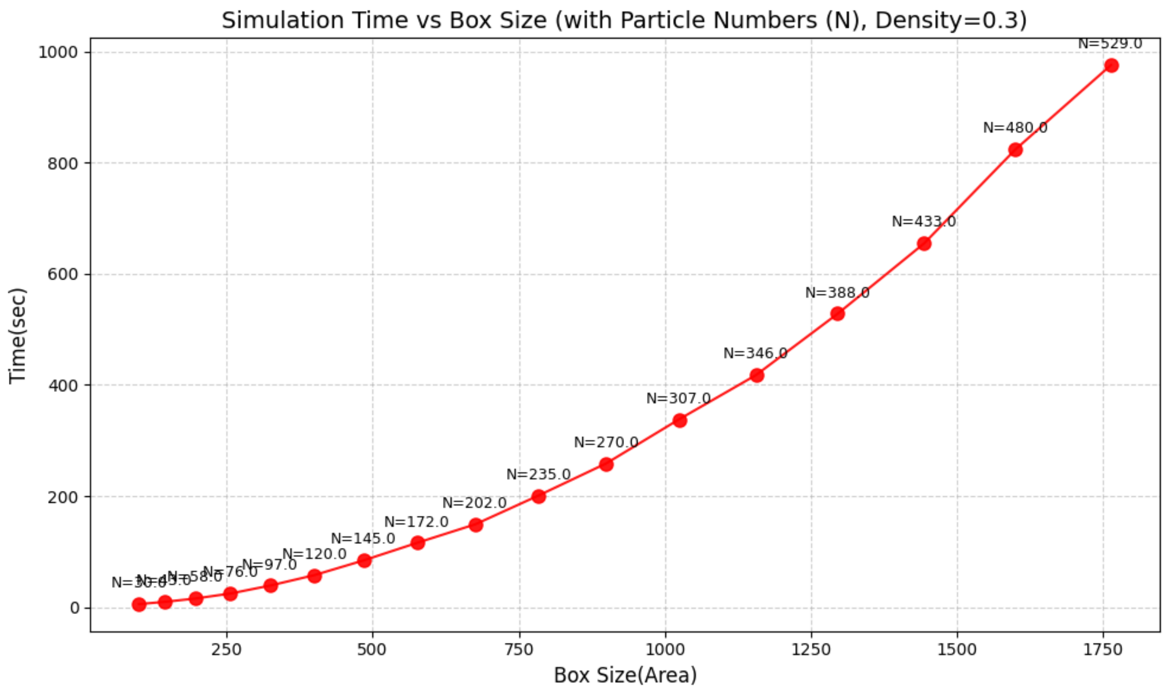

Figure 10: The simulation time in seconds required to simulate a box size of a given area. The number of particles scales linearly to keep the density constant.

To investigate the relationship between the simulation time and box size, we used 0.3 density with an initial configuration of box length equal to 10 and number of particles 30. As the box size grew, the number of particles grew accordingly to maintain the density of the system. Each simulation is 10^5 steps to make the simulation feasible for bigger system sizes. The results are in fig. 10. We can see clearly that the number of particles scales linearly with the box size to keep the density constant, as increasing just the dimension of the box with the number of particles constant is meaningless from a computational perspective. We can see here clearly that the simulation time scales as $O(N^2)$, (i.e., parabolically) with the system size as expected. This poses a challenge for simulating larger systems, and therefore, we have to come up with creative ways to manage this problem, as will be shown later.

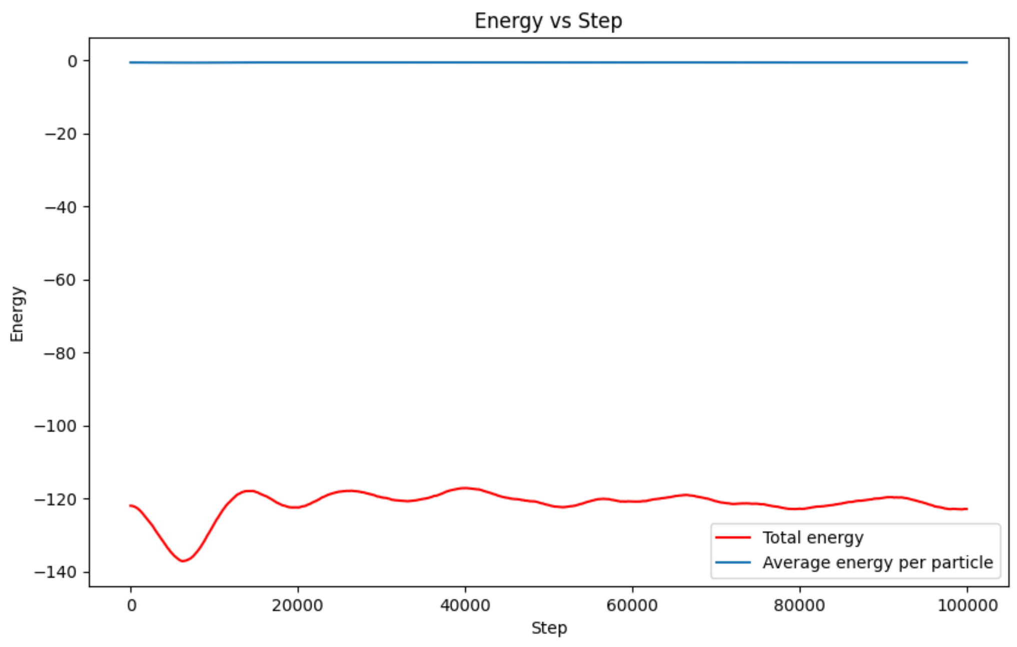

Figure 11: The total energy of the system and the average energy per particle for a simulation of a 3D box with 150 particles.

2.2.2 Three Dimensions

In the case of a 3D box, the highest density that could be achieved without the simulation crashing was 0.21 with an initial configuration as a cube lattice. A hexagonal packing as an initial configuration was also tried to make the simulation more stable but unfortunately, no considerable improvements were noticed besides delaying the crash.

A simulation was performed with a density of 0.21 with particles in a cube lattice as an initial starting point for the simulation. Random velocities in the range $[-0.4, 0.4]$ were generated and the simulation was run for 10^5 steps. Then a minimization procedure took place with a 10^7 maximum step and a minimum threshold for force equivalent to 10^{-4}. Also, the steepest descent algorithm will stop if the energy difference of two consecutive steps is less than 10^-7. The energy plot of the simulation is in fig. 11 and the energy of the system during minimization is in fig. 12. The final minimized structure can be found in fig. 13.

The simulation energy fluctuated much less than the 2D case with a constant average energy per particle through the whole simulation time. The minimization step was performed successfully, with a clear plateau of the energy curve before the minimization came to an end. The minimized configuration is in fig. 13, which shows a semi-hexagonal packed lattice form, which is an acceptable outcome as such energy minimizes the energy of the system and allows for close packing of the particles of equal size in the 3D space.

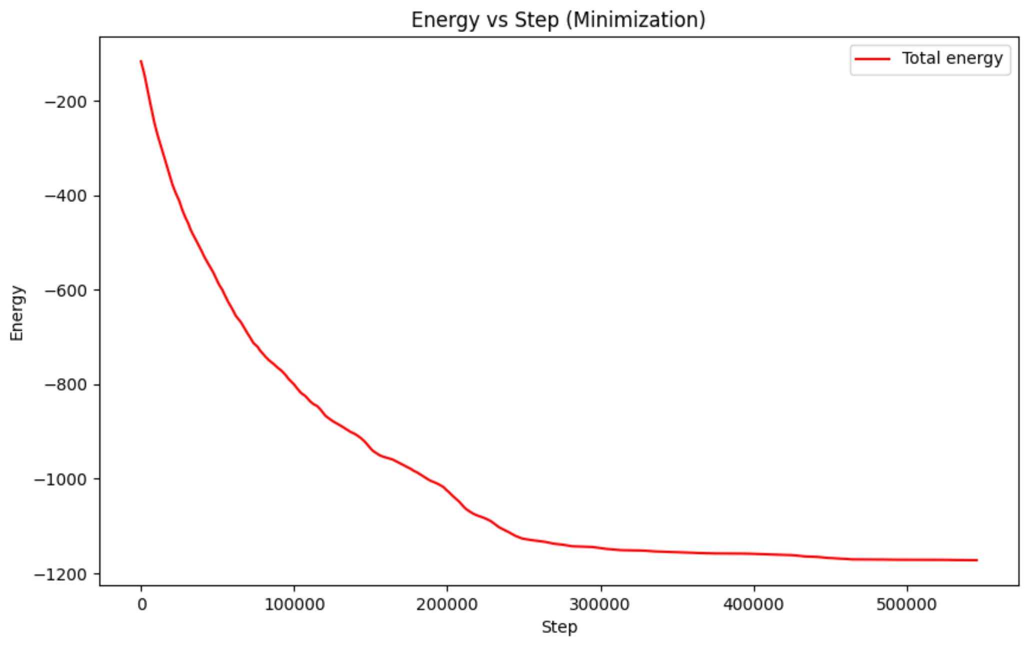

Figure 12: The energy of the system with 150 particles in a 3D box of length 10 during the minimization process by the steepest descend algorithm.

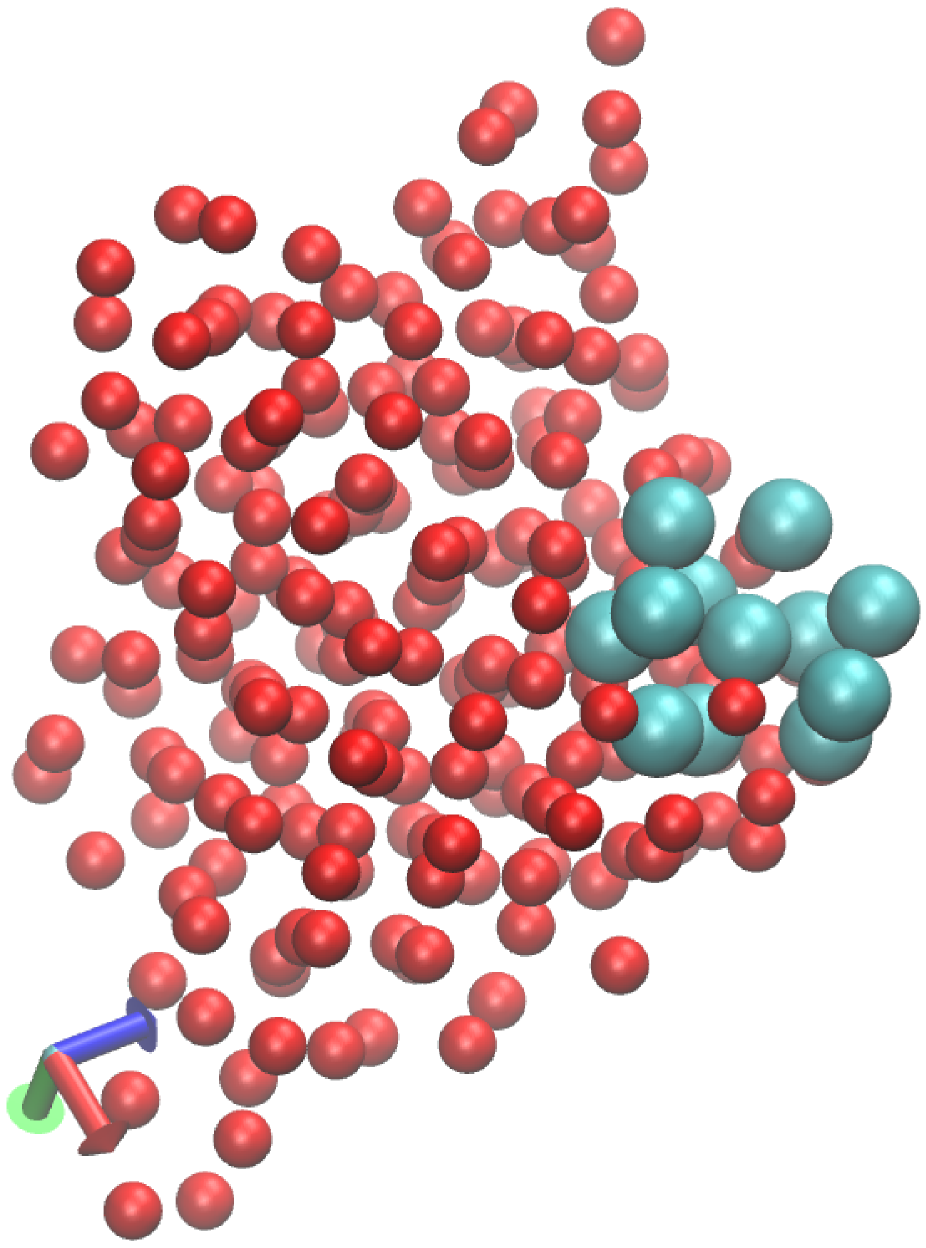

Figure 13: The final configuration of the system after minimization. Some particles are shown bigger with a different color (cyan) just for better visualization of the structure of the particles.

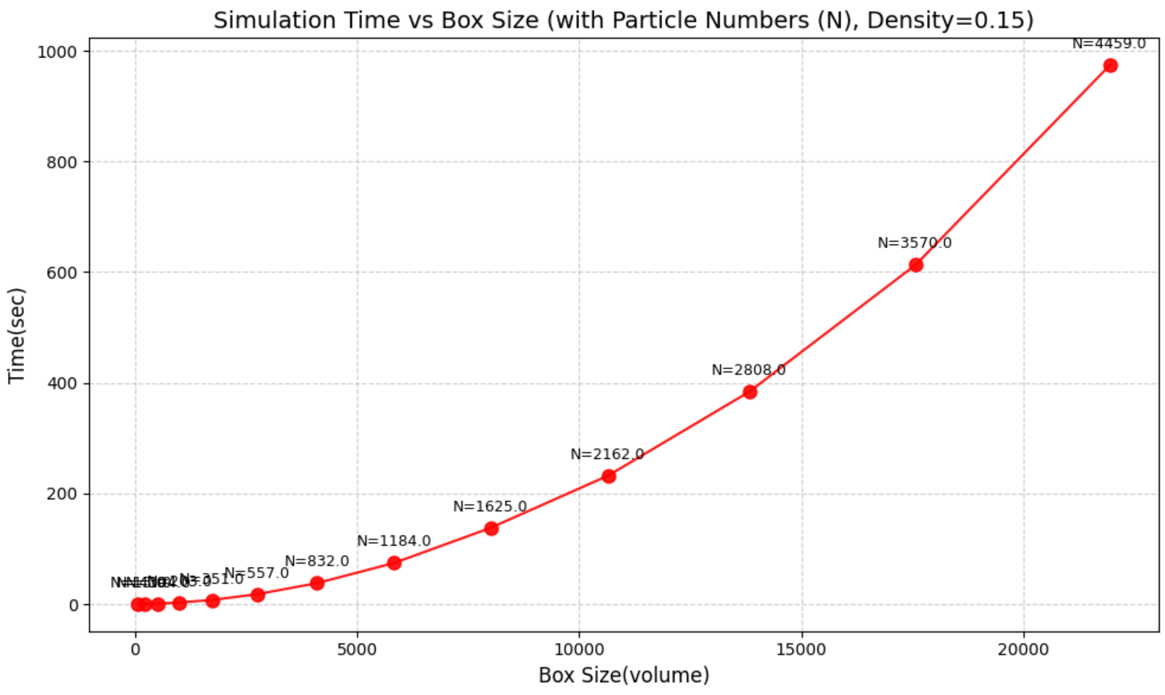

Figure 14: Simulation time of particles in a box interacting via Lennard-Jones potential for 3D.

Again, as in the 2D case, the simulation time was computed with different box sizes with a density of 0.2 kept constant. The initial system size was a 4 by 4 by 4 box with 13 particles in it. The results are in fig. 14. The parabolic scaling of the simulation time with system size is as expected, which, of course, poses a challenge for simulating large systems.

3 Part 3 and 4

3.1 Theory

Linked-cell algorithm (LCA) is a common method employed in MD simulation to decrease the computational cost of calculating short-range interactions. The algorithm works in the following steps:

1. Divide the space into smaller boxes with dimensions slightly larger than the cutoff distance such that the box length is divisible by the small box length.
2. Each particle is assigned to a cell based on its position and a linked list data structure is used to store particle indices in each cell.
3. For each cell, only particles in the same cell and half of the adjacent cells are considered for force calculations.
4. Compute the force between particle pairs.
5. Update the positions of the particles and repeat.

This reduces the computation from $O(N^2)$ to $O(N)$.

3.2 Method, Results and Discussion

3.2.1 Two Dimensions

We wrote a code implementing the linked cell algorithm for the calculation of LJ interactions. As a test, a simulation of 40 particles in a 10 by 10 box (density$=0.4$) was performed for 10^6 steps, and the positions of the particles were initialized in a triangular lattice configuration. The total energy (kinetic + potential) of the system through the entire simulation time can be seen in fig. 15. The energy of the simulation fluctuated around a constant value, and the energy per particle was stable throughout the simulation.

Next, a benchmark was performed to gauge the performance of the code and find how it scales with the increase in the system size. The starting configuration of the system was 30 particles in a 10 by 10 box, giving us a density of 0.3, which was maintained as the box size increased by scaling the number of particles accordingly. The results are in fig. 16. The results clearly show the linear relation between the simulation time and system size, proving the linear scaling of the linked cell algorithm. This shows that for large systems, the use of the linked cell algorithm is a must and it provides great benefit as it reduces the computational time of the simulation by almost 10 times as evident by comparing fig. 16 to fig. 10.

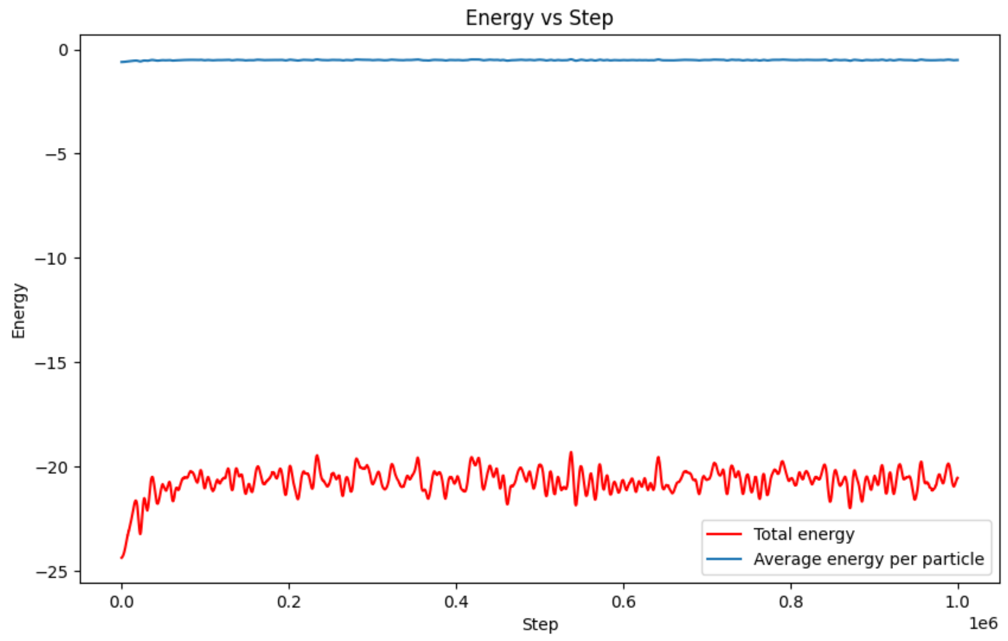

Figure 15: The total energy of the system and the average energy per particle of a 20 particle system in a 10 by 10 box using linked cell algorithm.

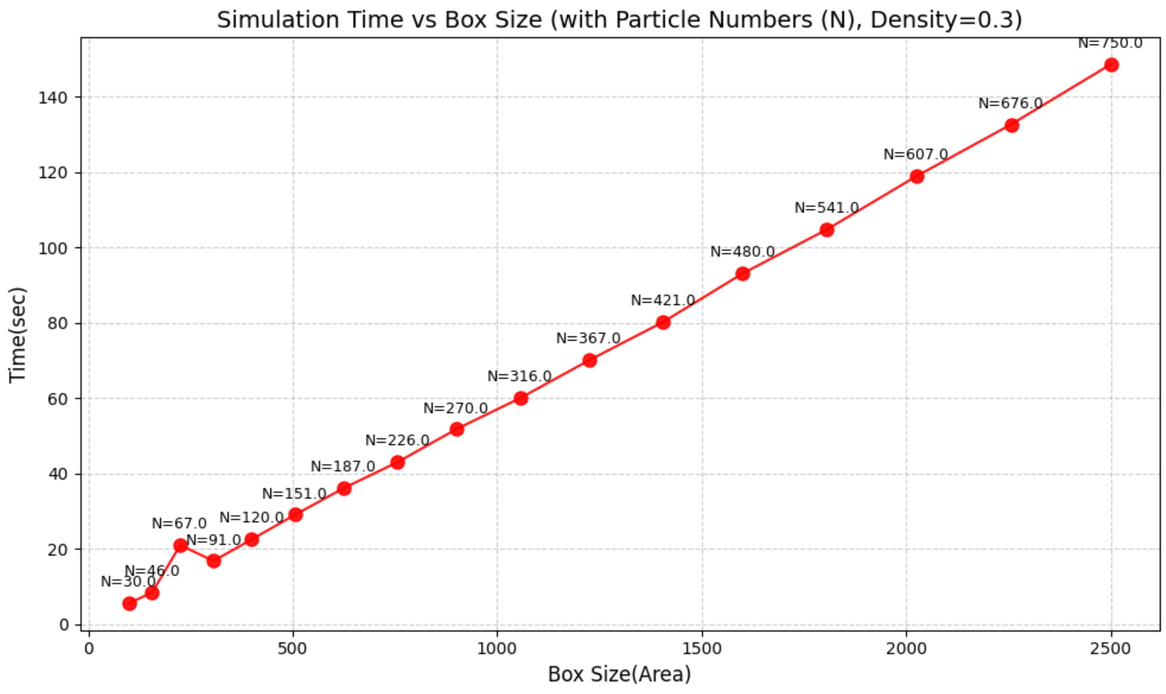

Figure 16: Computation time vs box size for a 2D system when using linked cell algorithm.

3.2.2 Three Dimensions

This time, the linked cell algorithm was extended to 3D, and a simulation of 210 particles in a 10 by 10 by 10 box was performed for 10^6 steps. The system's total energy can be seen in fig. 17.

Next, the computation time of the code was benchmarked against various system sizes, starting from a box of size 10 by 10 by 10 with 150 particles in it and then increasing the size gradually until reaching a box 34328.125 in volume with 5149 particles which took only 30.74 seconds to simulate for 10^4 steps. This is a massive improvement compared to the naive approach.

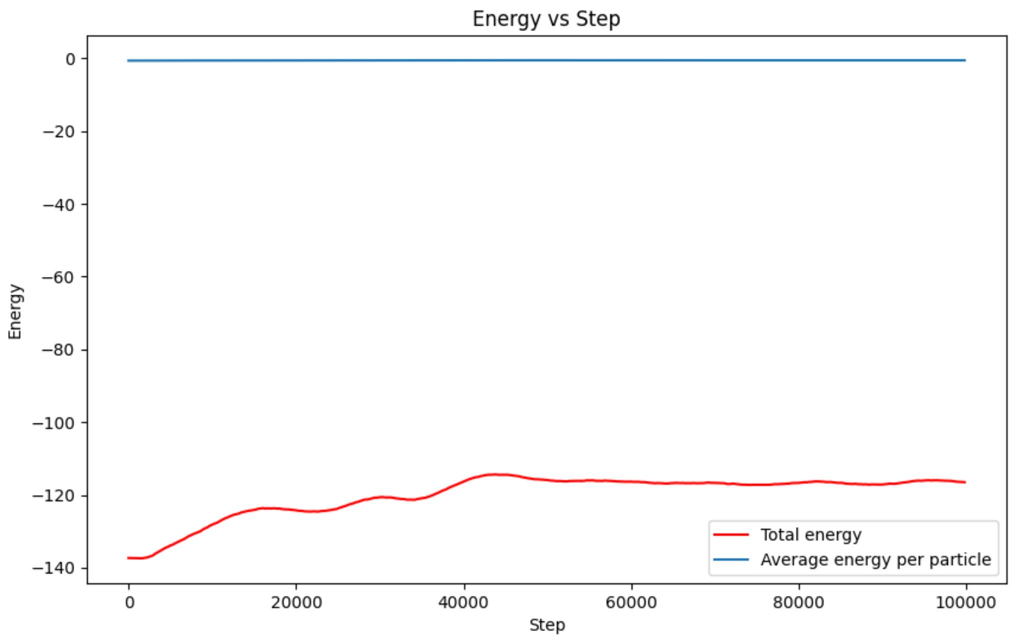

Figure 17: The total energy of the system and the average energy per particle of a 20 particle system in a 10 by 10 by 10 box using linked cell algorithm.

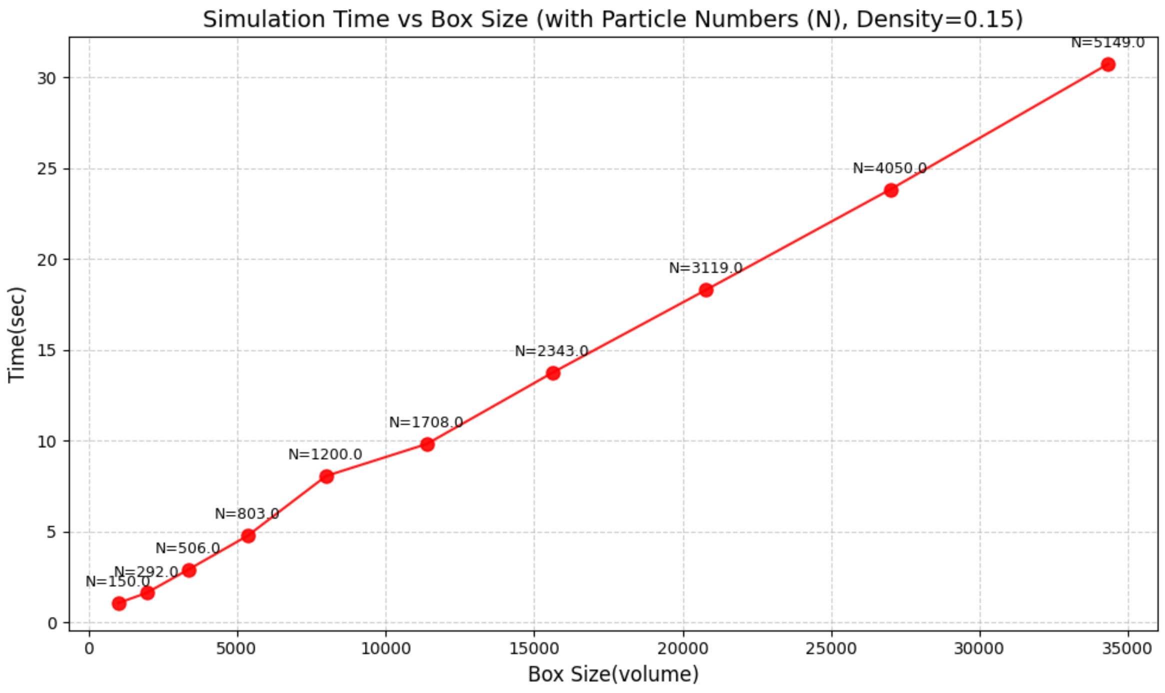

Figure 18: Computation time vs. box size for a 3D system when using linked cell algorithm.

4 Comparision and computational efficiency

From the theory and method section, we can predict that the linked cell algorithm reduces the time complexity of Big-O by a factor of N.
Both the 2D and 3D figures 19 20 compare simulation time vs. box area for naive LJ ($O(N^2)$) and linked-cell ($O(N)$) methods. The naive approach (green) shows quadratic scaling, becoming significantly slower for larger systems, while the linked-cell method (red) scales near-linearly, demonstrating superior efficiency for large particle counts. The graph shows that time complexity reduces from $O(N^2)$ of linked-cell algorithms.
From the 2D simulations graphs, we can also see that with a linked cell algorithm, we can increase the system area up to 2500 with 750 particles in 140 seconds. Also, from figure 10 and 16, we can see for a system area of 1600 with 480 particles, the linked cell algorithm takes $\approx$ 93 seconds, while the naive approach takes $\approx$ 824 seconds. Almost 9 times.
Similarly, for the 3D simulation, a comparable system of the volume of 625 takes $\approx$ 277 seconds for the naive approach, while the linked-cell algorithm only takes $\approx$ 14 seconds. Almost 20 times.
This gives us an idea of the power of the linked cell algorithm.
In addition, our code is written in `C++`, so it's already much faster than other commonly used programming languages like `Python`. We can increase the interval for the output of the trajectory to the file, But that will make our simulations look fast and jumpy, and we might miss something important in between two snapshots such as artifacts in the simulation. One idea that we considered was the parallelization of the code using MPI or OpenMP. However, as parallel link cell/ LJ simulation algorithms are different than the normal ones, we decided not to proceed with parallelization of our code.
Note: For high-density particle boxes, the linked cell algorithm sometimes crashes. As a result, we kept the density of both the 2D simulations at 0.3 and the 3D simulations at 0.15!

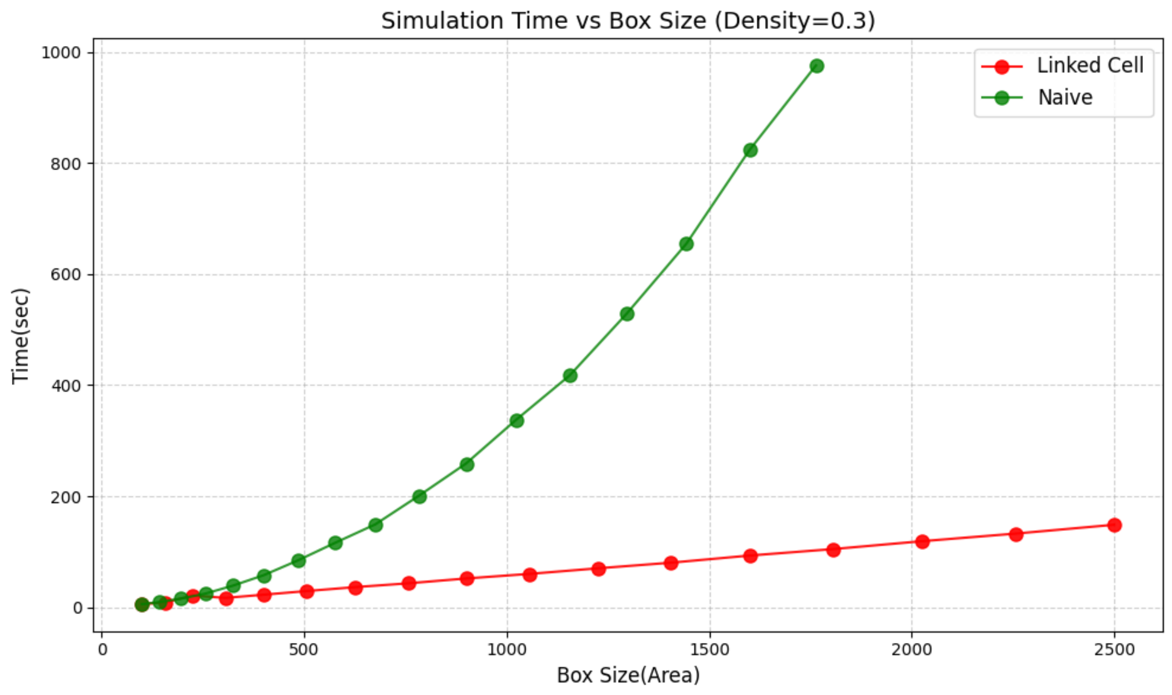

Figure 19: Simulation time vs. Box size for 2D Lennard Jones simulation.

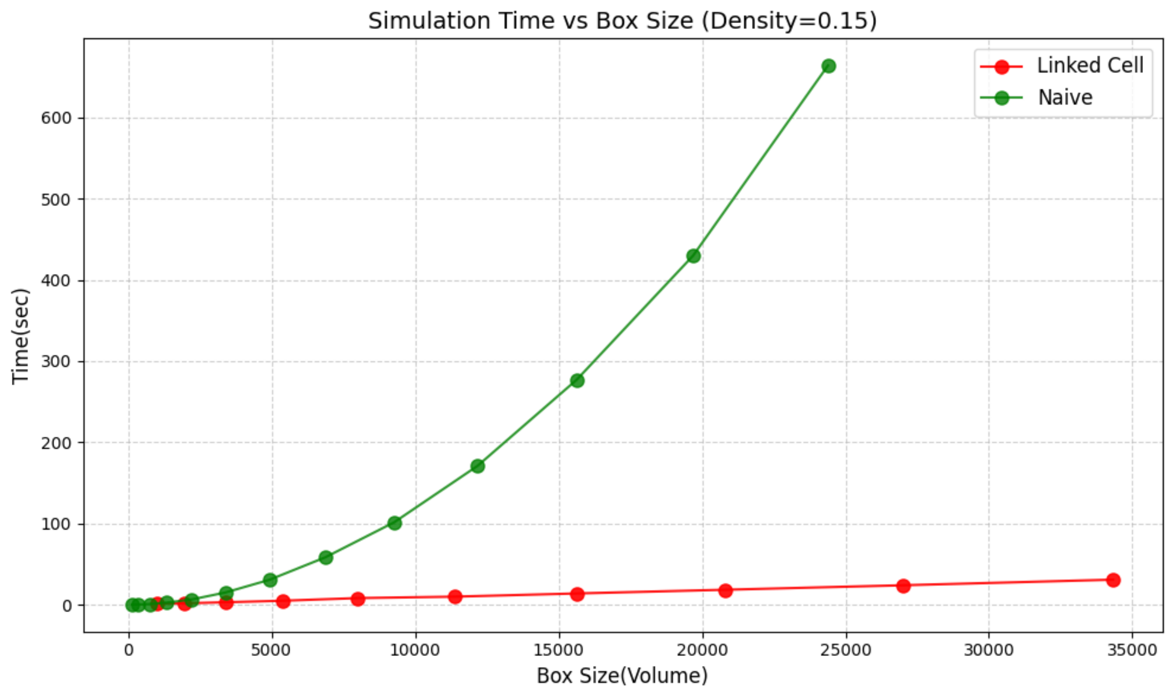

Figure 20: Simulation time vs. Box size for 3D Lennard Jones simulation.

Appendices

A Program Description

All the simulation codes are written in C++ except for the plotting parts. The project directory looks like this

```
bin               - contains binary programs after compiling
LICENSE           - LICENSE files.
makefile          - make file
README.md         - Brief README file
run               - Output run files.
scripts           - Plotting scripts
src               - Main source code
videos            - Output video. Converted to .mp4 using ffmpeg.
```

Usage Instruction

In the codes, we use **make** to compile and test the codes. All we need to do is to run

```
    make clean
        - This will clean all of your binary files.
    make
        - This will create all the binaries in the bin/  directories.
    make test
        - Will run all the simulations together.
```

If we want to clean all the run and binary files, we need to run **make cleanall** command.

Simulation Parameters

At the start of the code, there is a list of variables that control the simulation. Below here is a small description of how the code is controlled.

```
N              - number of particles (default: 40)
steps          - number of steps (default: 1e6)
dt             - time step (default: 1e-4)
L              - box length (default: 10.0)
rc             - Lennard-Jonnes cut-off distance (default: 2.5)
output_freq    - output frequency (default: 200)
lattice_type   - Lattice type initialization('t' for triangular, 's' for
   square) (default: 't')
cell_size      - per cell size
num_cells      - number of cells
total_cells    - total number of cells
```

Output

The result of this simulation is 4 files as follows:

```
Naive approach output:
    lj_<dim>_trajectory.xyz: this file contains the trajectory, which
       can be visualized.
    lj_<dim>_energy.dat: this is for outputting the energy for each
       trajectory step for plotting.
    lj_<dim>_min_config.xyz: This file contains the configuration after
       the minimization has finished.
    lj_<dim>_min_energy.dat: This file contains the energy of the
       minimized system.
Linked-Cell algorithm outputs:
    lj_cell_<dim>_trajectory.xyz: this file contains the trajectory,
       which can be visualized.
    lj_cell_<dim>_energy.dat: this is for outputting the energy for each
        trajectory step for plotting.
    lj_cell_<dim>_min_config.xyz: This file contains the configuration
       after the minimization has finished.
    lj_cell_<dim>_min_energy.dat: This file contains the energy of the
       minimized system.
```

A.1 Lennard-Jones 2D and Energy Minimization

Key Functions

- `initialize_positions()` – Places particles in a lattice. The supported lattice structure is triangle 't' and square 's'.
- `initialize_velocities()` – Sets initial velocities
- `calculate_forces()` – Computes LJ forces with PBC
- `run_simulation()` – Main MD simulation engine. Runs MD simulation
- `minimize_energy()` – Finds minimum energy configuration

A flow diagram of the key functions is given in fig. 21

A.2 Lennard-Jones 2D with linked cell algorithm

Key Functions

Core Simulation Functions

- `initialize_positions()` - Places particles in 2D lattice (triangular or square)
- `initialize_velocities()` - Sets random velocities with zero net momentum
- `build_cell_list()` - Organizes particles into spatial grid cells
- `calculate_forces_linked_cell()` - Computes LJ forces using neighbor cells
- `update_positions()` - Velocity Verlet position update with PBC
- `update_velocities()` - Velocity Verlet velocity update

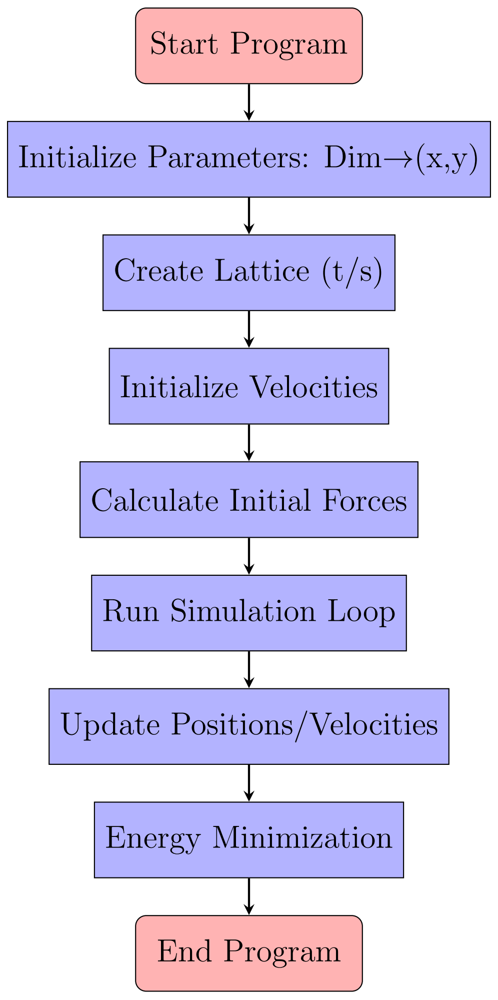

Figure 21: Flow diagram of the Lennard-Jones 2D simulation program

Physics Functions

- `distance()` - Minimum image distance with PBC
- `lj_potential()` - 12-6 Lennard-Jones potential
- `lj_force()` - LJ force calculation

A flow diagram of the figure is given in fig. 22.

A.3 Lennard-Jones 3D

Key Functions

Initialization

- `initialize_positions()` - Creates 3D lattice (cubic or hexagonal)

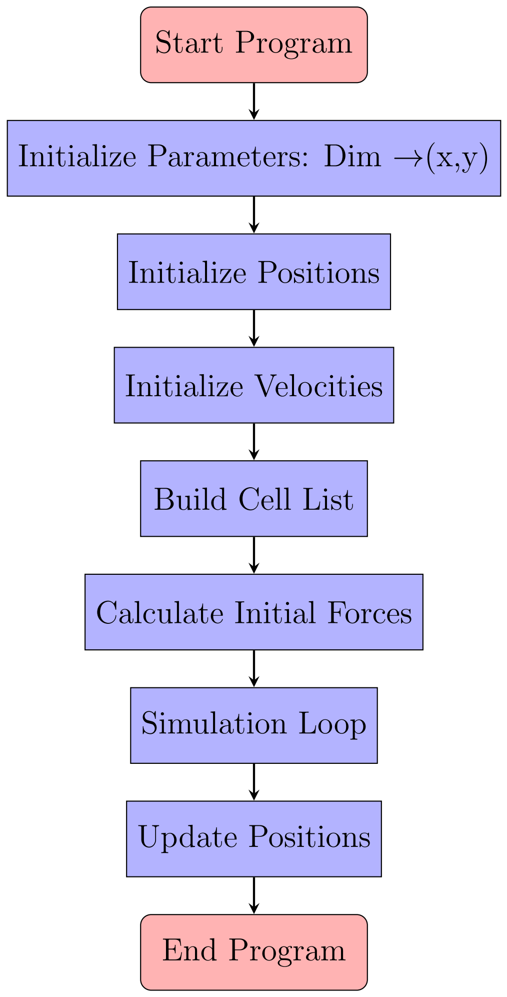

Figure 22: Flow diagram of the 2D Lennard-Jones simulation with linked cell algorithm

- `initialize_velocities()` - Sets velocities with zero center-of-mass

Core Simulation

- `calculate_forces()` - Computes LJ forces with PBC
- `update_positions()` - Velocity Verlet position update
- `update_velocities()` - Velocity Verlet velocity update

Energy Minimization

- `minimize_energy()` - Steepest descent minimization
- `calculate_forces_no_pbc()` - Force calculation without PBC

Physics

- `distance()` - 3D minimum image convention
- `lj_potential()` - 12-6 Lennard-Jones potential
- `lj_force()` - LJ force calculation

Below here is a diagram for Lennard-Jones 3d simulation fig. 23.

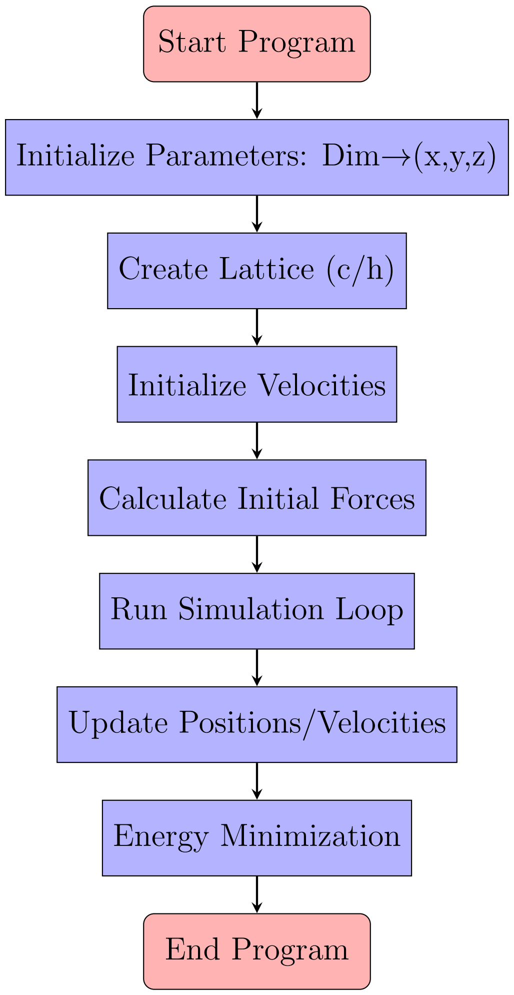

Figure 23: 3D Lennard-Jones simulation workflow with energy minimization

A.4 Lennard-Jones 3D with linked cell algorithm

This program simulates N particles in a 3D box with periodic boundary conditions using:

- Lennard-Jones potential with cutoff radius
- Linked cell algorithm for efficient neighbor searching and velocity Verlet integration

Key functions

The functions are listed below.

```
initialize_positions() - Places particles in 3D lattice
initialize_velocities() - Sets random velocities (zero net momentum)
build_cell_list() - Organizes particles into spatial grid cells
calculate_forces_linked_cell() - Computes LJ forces using neighbor cells
update_positions() - Moves particles (Velocity Verlet integration)
update_velocities() - Updates velocities (Velocity Verlet)
distance() - Minimum image distance with PBC
lj_potential() - 12-6 Lennard-Jones potential
lj_force() - LJ force calculation
run_simulation() - Manages simulation loop and I/O
```

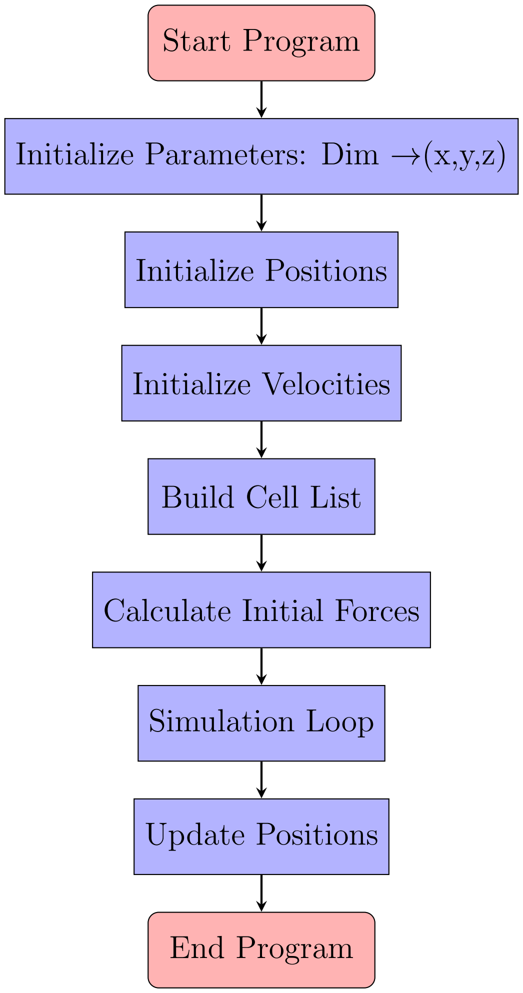

Figure 24: Flow diagram of the 3D Lennard-Jones simulation with linked cell algorithm