High-performance Computing

Coursework Assignment

Deadline: 21st March 2025 – 23:00

Instructions

Please take note of the following when completing this assignment:

- Read all the tasks carefully and plan ahead before you start.
- You may use any of the tools and libraries available on the provided Linux environment.
- Your submitted code **must** compile and run correctly on the provided Linux environment.

Make regular backups of your code onto a separate computer system; no allowance will be made for data loss resulting from human or computer error.

1 Introduction

The objective of this coursework is to write a parallel code for modelling basic molecular dynamics. For the purpose of keeping the physics simple, we will only add a minimal level of interactions between atoms. The focus is therefore on the efficient parallel numerical implementation of the algorithm and exploiting good software development practices, leveraging the techniques and tools learnt during the course.

2 Algorithm

Each particle in the system is assumed to be a point mass and has a position and velocity in three-dimensional space. Their motion is fundamentally governed by Newton's laws, satisfying the following first-order differential equations:

$$\frac{d\mathbf{r}_i}{dt} = \mathbf{v}_i,$$
$$\frac{d\mathbf{v}_i}{dt} = \mathbf{a}_i = \mathbf{F}_i/m_i,$$

where $\mathbf{F}$ captures the total force exerted on each atom.

We discretise in time using the first-order forward Euler scheme and a time-step of Δt, leading to the following update steps for each particle:

$$\mathbf{r}_i^{n+1} = \mathbf{r}_i^n + \Delta t\, \mathbf{v}_i,$$
$$\mathbf{v}_i^{n+1} = \mathbf{v}_i^n + \Delta t\, \mathbf{F}_i/m_i.$$

For the systems considered, a time step of $\Delta t = 0.001$ should remain stable.

We suppose our system is composed of only two species of particle:

- Type 0: mass $m_i = 1$;
- Type 1: mass $m_i = 10$.

For the test cases given later, the particle types are given explicitly. Otherwise, the first user-specified p% of particles should be of type 1, with the remainder being of type 0. The default choice should be 10%.

2.1 Lennard-Jones Potential

To model interaction between particles, we consider the inter-molecular forces experienced by each particle due to the relative proximity of other particles in the system. In particular, two atoms repel each other when they are very close to each other, but attract each other at moderate distances, with attraction dropping to zero at infinity. This behaviour is known to be well-modelled by the *Lennard-Jones Potential*, one form of which is given by:

$$\phi_{ij} = 4\epsilon_{ij}\left[\left(\frac{\sigma_{ij}}{r_{ij}}\right)^{12} - \left(\frac{\sigma_{ij}}{r_{ij}}\right)^{6}\right]$$

where r_{ij} is the distance between particles i and j, while σ_{ij} and ϵ_{ij} are parameters characterising the relative size and energy of the particles as a function of their type. For interactions between the combinations of the two types of particles in our system we use the following values:

ϵ	0	1
0	3.0	15.0
1	15.0	60.0

σ	0	1
0	1.0	2.0
1	2.0	3.0

The total force exerted on particle i is then given by

$$\mathbf{F}_i = -\nabla \sum_{j \neq i}^{N} \phi_{ij}.$$

To compute the above, we need the derivative of the Lennard-Jones potential, with respect to each of the coordinate directions. The case for x is given by:

$$\frac{d\phi_{ij}}{dx} = -24\epsilon x_{ij}\left(\frac{2\sigma^{12}}{r_{ij}^{14}} - \frac{\sigma^{6}}{r_{ij}^{8}}\right),$$

where $x_{ij} = x_j - x_i$. The derivatives for the other coordinate directions follow similarly.

2.2 Domain and Boundary Conditions

The domain, D, of our particle system is a fixed box, defined as:

$$D = [0, L_x] \times [0, L_y] \times [0, L_z].$$

When particles reach a boundary, they are reflected back into the domain. For the x direction, we have

$$\left.\begin{array}{r} x_i = -x_i \\ u_{i,x} = |u_{i,x}| \end{array}\right\} \quad x < 0,$$
$$\left.\begin{array}{r} x_i = 2L_x - x_i \\ u_{i,x} = -|u_{i,x}| \end{array}\right\} \quad x > L_x,$$

with similar conditions for the y and z coordinates. Here, we use $u_{i,x}$ to denote the x-component of the velocity.

2.3 Initial Conditions

In general, the particle system is composed of N particles, initially distributed randomly within the domain, and each with a random initial velocity.

Particles should be inserted into the system, one-by-one, with each component of position being uniformly sampled from the domain length in the corresponding direction. To avoid instability, particles should only be inserted if they are at least a distance $R = 0.5$ from all other particles. If they are too close, they should be discarded and a new particle sampled.

For initial particle motion, each component of velocity should be sampled from the interval $[-0.5, 0.5]$.

In addition to random particle insertion, specific configurations may also be hard-coded, as given in the test cases (Section 3).

2.4 Kinetic energy

The kinetic energy of the system is given by the usual formula:

$$E = \frac{1}{2}\sum_i m_i |\mathbf{u}_i|_2^2$$

2.4.1 Setting the temperature

The temperature of the system is proportional to the average kinetic energy of the particles and is given by:

$$T = \frac{2}{3k_b}E$$

where $k_b = 0.8314459920816467$ is the Boltzmann constant (expressed in units of Daltons $\times$ Angstroms squared per picosecond squared per Kelvin).

The temperature can be optionally fixed to T_0 by rescaling the velocities by a constant $\lambda = \sqrt{T_0/T}$, i.e.

$$\mathbf{v}_i \leftarrow \lambda \mathbf{v}_i$$

3 Test cases

Assuming a domain of size $20 \times 20 \times 20$, the following can be used to test your code. For each of these cases, you should not impose a temperature.

Test case	T	Particle	x	y	z	u	v	w	type
1	1.0	1	10	10	10	0.0	0.0	0.0	0
2	20.0	1	10	10	10	5.0	2.0	1.0	0
3	50.0	1	8.5	10	10	0.0	0.0	0.0	0
		2	11.5	10	10	0.0	0.0	0.0	0
4	50.0	1	8.5	11.5	10	0.5	0.0	0.0	0
		2	11.5	8.5	10	-0.5	0.0	0.0	0
5	50.0	1	8.5	11.3	10	0.5	0.0	0.0	0
		2	11.5	8.7	10	-0.5	0.0	0.0	0
6	50.0	1	8.5	11.3	10	0.5	0.0	0.0	1
		2	11.5	8.7	10	-0.5	0.0	0.0	1

The qualitative behaviour of these cases should be as follows:

1. Particle should remain stationary indefinitely.
2. Particle should bounce around the box.

3. Two particles should start stationary and gradually move towards each other, before being repelled. The cycle should repeat indefinitely. Neither particle should come into contact with the boundary.
4. Two particles pass each other, causing them to be slightly deflected. After rebounding off the side walls, they should subsequently collide.
5. Two particles pass each other more closely, causing them to encircle each other, collide once and move off towards the side boundaries.
6. Two particles oscillate towards and away from each other, while slowly rotating around the centre of the domain.

4 Code usage

Your code should accept all configuration options as arguments on the command-line. The user should not need to interact with the program after it is run. The list of options your code should accept are given below and should be displayed if the user provides the `--help` command-line argument:

```
$ ./md --help
Allowed options:
--help                    Print available options.
--Lx arg (=20)            x length (Angstroms)
--Ly arg (=20)            y length (Angstroms)
--Lz arg (=20)            z length (Angstroms)
--dt arg (=0.001)         Time-step
--T arg                   Final time
--ic-one                  Initial condition: one stationary particle
--ic-one-vel              Initial condition: one moving particle
--ic-two                  Initial condition: two bouncing particles
--ic-two-pass1            Initial condition: two passing particles
--ic-two-pass2            Initial condition: two passing particles close
--ic-two-pass3            Initial condition: two passing particles close, heavy
--ic-random               Initial condition: N random particles
--percent-type1 arg (=10) Percentage of type 1 particles with random IC
--N arg                   Number of particles to spawn with random IC
--temp arg                Temperature (degrees Kelvin)
```

- If the `--help` option is provided, the above is printed and the program exits.
- It is expected that only one option beginning with `--ic` is provided.
- The options `--N` and `--percent-type1` are required only if the `--ic-random` option is provided.
- The argument `--temp` allows the user to optionally fix the temperature as described in Section 2.4.1.
- All other options are mandatory.

For the test cases only, your code should output two text files

1. Particle data with columns of data giving – one row for each particle – the time, particle number, position and velocity.
2. Kinetic energy data with columns of data giving the time and kinetic energy.

Data should be output every 0.1 time units. e.g. for a simulation of 10 time units, there should be 101 rows in the kinetic energy file.

For the general case with random particle distributions, your code should only store the kinetic energy file.

Continues on next page ...

Tasks

Please make sure to read all the tasks below before starting to write code, to help steer your code design.

Write C++ codes which implements the molecular dynamics model described above, following the instructions below. Write a report (maximum 5 pages) as described in the tasks.

1. Create a new Git repository and continue to use git appropriately throughout your code development.

   When submitting, provide evidence of your use of git by running the command

   ```
   git log --name-status > repository.log
   ```

   and including the file `repository.log` in your submission. [5%]

2. Write a serial code which implements the molecular dynamics algorithm above. [20%]

   Report: For all cases, plot the kinetic energy as a function of time. For test cases 3-6, also plot the trajectories on the x-y plane. [5%]

3. Write a Makefile or CMake configuration for your project. Use the target `md` to compile your serial code. Include suitable `default` and `clean` targets. [5%]

4. Write tests for your code which execute the test cases and verify the correct particle positions and kinetic energy at the end times are obtained to within a sensible tolerance. Add a target `unittests` to your build system configuration which runs the unit tests. [5%]

5. Optimise your serial code using a profiler. Check your optimised code still passes the tests.

   Report: Describe, with quantitative evidence, the most time-consuming parts of your code. List optimisations you have made to the code to improve performance, and quantify the change observed. [10%]

6. Add support for generating HTML code documentation using doxygen and document the source code appropriately. Add a `doc` target to the Makefile to compile the documentation. [5%]

7. Extend the serial code to add shared-memory parallelism. Update your Makefile appropriately to compile the target `mdpar` for parallel execution. [10%]

   Report: Provide a plot showing the speed-up of your code when run with various levels of parallelism up to 48 cores. To do this, you should submit jobs to the cluster. Make sure to use a large enough test case, e.g. 10,000 particles in a $50 \times 50 \times 50$ domain. Include the SLURM script `job-script.slr` used to run your code on 48 cores in your submission. [5%]

   Report: Describe your approach to adding shared-memory parallelism and any considerations you gave to maximise performance. Describe how you measured the timings above [5%]

8. Extend the serial code to off-load the computationally intensive parts to a GPU. For this part, you will need to use `typhoon.ae.ic.ac.uk`. [10%]

   Report: Describe your CUDA implementation and explain the reasons behind the approach you took. [5%]

9. Suppose we approximate the Lennard-Jones potential to only be effective up to a distance R, and particles can be considered to have no interaction beyond this distance.

   You do not need to actually change your code in this task.

   Report: Describe how you might refactor your code to support distributed parallelism in this case. Include how you might distribute the computation, how the different parts of the problem would be coordinated to produce the same result as the serial code. What would be the challenges faced for this problem in order to allow it to scale to large numbers of processors? [5%]

10. Use good coding practices (code layout, comments, etc) throughout your code. [5%]

Continues on next page ...

Submission and Assessment

When submitting your assignment, make sure you include the following:

- All the files needed to compile and run your C++ codes which solves the molecular dynamics problem, as described above. i.e. All `.cpp` and `.h` files necessary to compile and run the code.
- The git log (`repository.log`) from Task 1.
- The `Makefile` or `CMakeLists.txt` file from Task 3.
- Any files specific to the tests included in Task 4
- The doxygen configuration from Task 6 (not the generated HTML files).
- The SLURM script used in Task 7.
- Your report (maximum 5 pages, in PDF format only).

These files should be submitted in a **single `ZIP` archive file** to Blackboard Learn.

It is your responsibility to ensure all necessary files are submitted.

You may make unlimited submissions and the last submission before the deadline will be assessed.

END OF ASSIGNMENT