Assignment - N-body Simulation

Parallel Scientific Computing I

Anne Reinarz, Max Fasi

Hand-out date: November 2, 2021

This coursework is to be completed via GitHub Classroom. You should accept the assignment via the URL https://classroom.github.com/a/V50mrDrP and add your name and CIS ID at the top of the file `README.md`.

Submission details

- Hand-in date: 21 February 2022
- Submission mode: Github Classroom/Blackboard Learn Ultra

Each section of this coursework consists of an implementational component and some questions. The questions should be answered in a short report (2 pages maximum), to be added to the repository.

Step 1: Multiple free objects

The first task is to extend `assignment-code.tex` into an N-body simulator. This code will be the starting point for the optimisations and parallelisation in later sections.
In the template code given to you, one object (the first one) is free, while all the other ($N-1$) bodies are fixed in space. Alter the code so that all objects move freely through space. Furthermore, make two objects fuse into one if they collide, and ensure that the final output of the code (minimal object distances, maximum velocities, final number of objects and objects' position) remains valid.
Two objects b_1 and b_2 collide if after a time step they end up in positions x_1 and x_2 such that

$$|x_1 - x_2| \leq C(m_1 + m_2), \qquad C = 10^{-2}/N,$$

where m_1 and m_2 are the masses of b_1 and b_2, respectively, and N is the number of bodies at the beginning of the simulation. It is not necessary to check the trajectory of the objects, you can simply use their position at the end of each step. The merger of the two objects has mass

$$m = m_1 + m_2.$$

Velocity and position should be the mass-weighted mean of the velocities and positions of the colliding objects: the new object should have position

$$x = \frac{m_1 x_1 + m_2 x_2}{m_1 + m_2}$$

and velocity

$$v = \frac{m_1 v_1 + m_2 v_2}{m_1 + m_2},$$

where v_1 and v_2 are the velocities of b_1 and b_2, respectively.
Marks will be given for the correctness of the solution. The solution to this step is to be implemented in a file called `step-1.cpp`. There is a Python script that allows you to generate arbitrarily complex, random initial configurations of particles. We will use this same script to generate scenarios. The masses $m_1, \ldots, m_N$ will be normalised so that $m_i \in]0, 1]$.

Questions

- What is the complexity of your implementation in terms of N, the number of bodies at the beginning of the simulation? Why?
- What methods could be used to reduce the complexity of the implementation? Note: you do not need to implement these improvements.
- What is the convergence behaviour of the implemented time-stepping scheme? Create a plot which clearly shows the convergence order. Which order is it? Is it the order you expected? Why?

  For this, we recommend that you take a two-particle setup and you track where the particles collide. Rely solely on numerical data, i.e., do not try to use any analytical solution.

Criterion	**Marks**	**Comment**
- correct movement	6	
- fusion	6	
- speed of implementation	8	
- questions	5	
Total	25	

Step 2: Short-range Forces

Now assume that the force acting on the bodies is no longer gravity, but some short range force such as Lennard-Jones:

$$F_{ij}(r_{ij}) = \begin{cases} \left(|r_{ij}|^{-12} - |r_{ij}|^{-6}\right) \frac{r_{ij}}{{r_{ij}}^2}, & \text{for } r_{ij} \le r_c, \\ 0, & \text{for } r_{ij} > r_c, \end{cases} \tag{1}$$

with a cut-off radius r_c. Modify the code from step 1 to use this force instead. Implement the method in a file called `step-2.cpp`. You may drop the code handling the collision of objects. Make sure to choose a reasonable value for the cut-off radius r_c.
Using the existing data structures will lead to an inefficient code. In the lecture we saw a linked cell method used for this case. For full marks modify the existing data structures to efficiently find all bodies in the cut-off radius.

Questions

- Describe the data structure you have chosen to implement the method.
- Would you change your choice if the particles were moving very quickly? Or if they were only vibrating?

Criterion	**Marks**	**Comment**
- data structures	5	
- correctness	5	the cut-off has not introduced a large error
- obtained efficiency	10	
- questions	5	
Total	25	

Step 3: Vectorisation

This step extends the source code from step 1 and makes the code exploit vectorisation. Assess the efficiency of your solution. Where appropriate, add OpenMP single-thread vectorisation pragmas. You might want to use performance analysis tools to find the appropriate places.
Marks will mainly be awarded for performance, but you have to ensure that you don't break the code semantics. The solution to this step is to be implemented in a single file called `step-3.cpp`.

Criterion	**Marks**	**Comment**
- correctness	5	The results of step 1 have not changed through parallelisation
- obtained efficiency	15	
Total	20	

Step 4: Instruction-level parallelism

This step should extend the source code from step 3 by adding another level of parallelism. Parallelise your code with OpenMP multi-thread parallelisation pragmas.
Ensure that you do not break the global statistics v_{max} and dx_{min} which are plotted after each step. This step assesses your understanding of the OpenMP parallelisation paradigm. Marks will be awarded for performance, but you have to ensure that you don't break the code semantics. The solution to this step is to be implemented in a single file called `step-4.cpp`.

Questions

In this section you will assess the performance of your code. You can either use your own machine (if reasonably powerful) or use Durham's supercomputer Hamilton for this section.

- Provide some background information about the machine you are using (even if it is Hamilton).

- Study the scalability of your code and compare it to a strong scaling model. Create a plot showing the scaling of your code.

  Note: This question can be completed using step 1, if you have struggled with step 3 and 4. Mark this clearly in your report.

These questions assesses your understanding of how to design numerical and upscaling experiments, how to present findings and how to calibrate models. Marks will be awarded for the data and the presentation.

Criterion	**Marks**	**Comment**
- correctness	5	The results of step 1 have not changed through parallelisation
- obtained efficiency	15	
- questions	10	
Total	30	

Using Hamilton

If you plan to use Hamilton, please start early. If you submit tests to run on the machine, these tests are queued—you share the resources with others—so you will have to wait a few hours before they start. Make your runs as brief as possible: this way, the scheduler of the supercomputer will squeeze them in as soon as a node is idle. For the speedup studies, a few minutes of simulation time are typically sufficient.

Important submission requirements

You should use the code framework in `assignment-code.cpp` as your starting point.

- In this repository we will expect to find files called `step-1.cpp`, `step-2.cpp`, `step-3.cpp` and `step-4.cpp`, plus one PDF file `report.pdf`. If you don't stick to the naming conventions, your submission will not be marked. There is a shell script which checks that you have the correct files in your repository. You can run it with `./validate.sh`. It is also run automatically by GitHub CI every time you push code to the repository.

- A Python validation script is run via GitHub CI every time you push code to the repository. The script will tell you whether your files conform to the submission guidelines and will run some basic tests. You can run also run the validation script manually via `python3 validate.py`.

- Make sure that the code does not produce any additional output in the terminal, i.e., do not alter any screen output. If you make the code accept any additional arguments, make sure that they are optional.

- In the report write down what you observe. If you know your implementation is wrong, tell us what you observe, and what you would have expected instead: you can still achieve full marks for that step if the bug in your code is trivial.

- In order to mark it, we will compile your code on a Linux system using the compilers `g++` (with flags `-fopenmp -O3 -march=native --std=c++0x`) and `icpc` (with flags `-fopenmp -O3 -xHost --std=c++0x`) . We recommend that you test your submission on Hamilton with these commands. You can do this automatically by using the `Makefile` included in this repository.
- If you are having trouble visualising the output with ParaView, make sure that you have switched to GaussianPoint in the properties tab.
- On Blackboard Learn Ultra, submit a single text file containing only:
  - the commit hash of your final submission, and
  - your GitHub username.

Collaboration policy

You can discuss your work with anyone, but you must avoid collusion and plagiarism. Your work will be assessed for collusion and plagiarism through plagiarism detection tools.