Durham University
Department of Computer Science

Coursework: Particles & Gravity

Computing Methodologies III: Numerical Algorithms

Computing Methodologies III: Parallel Programming

Tobias Weinzierl

October 2, 2019

This assignment is to be completed and handed in via DUO. For deadlines, please consult DUO and the level handbook. This is a joint assessment for the two submodules Numerical Algorithms and Parallel Programming.

Instructions

- This assignment is to be completed using the assignment's source code provided via DUO. It simulates three free objects in space (think of a planet or a satellite) where one is moving around the others through gravity.
- The only routine that you have to change is `void updateBody()`, though you might decide to alter other parts of the code as well. However, we will only mark `updateBody()`, i.e. ensure all the critical code that you want to be marked is found in there. Do not substructure this routine further.
- The code is not to use any other parameters than those currently passed through the command line, and you are not allowed to make your submitted code output any additional information to the terminal. This does **not** mean that it does not make sense to add additional outputs for your own experiments (to extract additional helper information about particular object positions, e.g.), but ensure that you strip your submitted code off any of such additional outputs.
- For each assignment step, you are expected to submit particular artefacts. It is either a piece of source code (do not submit multiple files, but keep the structure; ensure that the code continues to compile exactly the same way the given template does) or a PDF document. Ensure all page limitations are observed and submit PDFs only. Word files, e.g., are not accepted. Stick exactly to the submission format.

Step 1: Multiple free objects

In the template code given to you, one object (the first one specified on the command line) is free, while all the other molecules are fixed in space. Alter the code such that all objects move freely through space. Ensure that the global statistics (minimal distance between all objects and maximum velocity) are still computed correctly. Marks will be given for the correctness and also for the efficiency of the implementation (try to spot redundant calculations). The solution to this step is to be handed in as a single file called `solution-step1.c`.

This step is worth 25 marks for Numerical Algorithms.

Step 2: Pythagorean three-body problem

Given is the following experiment: Three objects are simulated. They have the masses $m_1 = 3, m_2 = 4$ and $m_3 = 5$. Each has a diameter of 10^{-2}. They are arranged at the vertices of a triangle with edge lengths 3,4 and 5. The object with mass 5 is placed opposite the face with length 5, the object with mass 4 is placed opposite the face with length 4, the object with mass 3 is placed opposite the shortest face. The initial velocity of all objects equals 0. This corresponds to the following input parameters:

```
./assignment 0.01 100.0 1e-8 0 0 0 0 0 0 4 3 0 0 0 0 0 5 3 4 0 0 0 0 3
```

Whenever two objects i and j collide, they fuse into one (something alike black hole fusion)[1]. The result object has a mass that is equal to $m_i + m_j$, and each component of its velocity is given as follows:

$$v = \frac{m_i}{m_i + m_j} v_i + \frac{m_j}{m_i + m_j} v_j.$$

Make your code terminate at the very moment when there's only one object left, and write out the position of this last object as `x, y, z`.

Submit your code as `solution-step2.c` and also submit a report on your approach plus your results. The latter is to be handed in as a one-page PDF called `solution-step2.pdf`. Address the following questions in your report:

- How does the position of the points depend on the time step size chosen, i.e. can you observe that the position becomes more accurate (convergence) if you use a finer time step size?
- Can you derive the convergence order of the method experimentally using the collision points?

Here are some hints:

1. Track the collision points for various time step sizes. Compute the distances between any two collision points for different time step sizes. Plot the data. Hint: Log-log plots help.
2. Find a correlation between time step size and change of the position of the collision point. How does the distance between two collision points change if you reduce the time step size? Do not only draw a line through your graph but compute a function that describes the experimental behaviour.

This step is worth 50 marks for Numerical Algorithms.

[1] The "original" Pythagorean three-body problem as you find it in literature works without a collision model and studies very complex trajectories. Without an object fusion model, these trajectories are quite ill-posed, i.e. minor changes of the initial object position yield dramatically different solutions. When you prepare for the exam, it might be worth revisiting this piece of coursework and play around with this phenomenon.

Step 3: Buckets of objects

Run your setup for hundreds or thousands of objects. You will observe that the simulation runs for quite some time. Sort the objects according to their velocity after every time step into buckets: The slowest objects with velocity $0 \leq v < v_{bucket}$ end up in the first bucket, the objects with $v_{bucket} \leq v < 2v_{bucket}$ in the second, and so forth. v_{bucket} depends on the maximal velocity and your choice of buckets (start to debug with only two of them). One time step now runs one Euler step with time step size ΔT on all objects in the first bucket (the slow guys). After that, it runs two time steps with time step size $\Delta T/2$ on the objects in the second bucket, four with $\Delta/4$ on the next one, and so forth.

The solution to this step is to be handed in as a single file called `solution-step3.c` and it shall be hard-coded to use 10 buckets. This step is worth 25 marks for Numerical Algorithms.

Step 4: Parallelisation with OpenMP

For this step, you are allowed to "roll back" to the solution of Step 1, i.e. the gravity model where all objects are allowed to move freely. Inefficiencies in the solution to Step 1 will not have any knock on effect from hereon. Parallelise your code with OpenMP. To assess the quality of your parallelisation, I recommend that you run experiments with a lot of objects and that you use tool support (correctness and analysis) where appropriate.

The solution to this step is to be submitted as `solution-step4.c`. The step is worth 50 marks for Parallel Programming.

Step 5: Experiments

Study the scalability of your code and compare it to both a weak scaling and a strong scaling model. Which one is more appropriate and what are the involved constants? Hint: Switch off all file output for your experiments.

For this step, you can either use your own machine (if it is reasonably powerful) or use Durham's supercomputer Hamilton. In both cases, document the spec of the machine used. If you plan to use Hamilton, here are some hints:

- Start early. If you submit tests to run on the machine, these tests are queued—you share the resources with others—so you have to expect a few hours until they start.
- Familiarise yourself with the machine early. To use a supercomputer, you have to write some startup scripts, you have to familiarise yourself with a modules environment, and so forth. It is not a lot of time to do this, but sometimes you have to wait (see above) for follow-up steps.
- Make your supercomputer runs as brief as possible. This way, the scheduler on the supercomputer will squeeze them in whenever some idle time arises on a node. For the numerical studies, you will have to run your code for a while. For the speedup studies, a few seconds/timesteps typically do the job.

Your findings are to be submitted as a one-page PDF report `solution-step5.pdf`. This step is worth 25 marks for Parallel Programming.

Step 6: Advanced parallelisation

Parallelise the solution to Step 3 with OpenMP. This in particular means you have to parallelise the sorting.

The solution to this step is to be submitted as `solution-step6.c`. The step is worth 25 marks for Parallel Programming.