Durham University
School of Engineering and Computing Sciences

Assignment: Space Bodies

Computing Methodologies III – Numerical Algorithms
Computing Methodologies III – Parallel Programming

This assignment is to be completed and handed via DUO. For an exact deadline consult DUO. This is a joint assessment of two submodules.

Assessment outline
You are to implement a very simple multibody simulation. Marks will be awarded for the implementation, for the numerical experiments, the discussion of the implementation as well as the results, the parallelisation approach, the discussion of this parallelisation, and the parallelisation outcome (efficiency). A code template/starting point for the assignment is given in C/C++ on DUO. The equivalent code is also provided in Python. Yet, the C/C++ code is to be extended. Students are strongly encouraged to augment their report with videos illustrating the code behaviour/numerical results, to upload these videos to their University webspace, YouTube or any other platform, and to add links to these videos to their report.

Step 1 – a multibody galaxy simulation
The code fragment on DUO studies an arbitrary number of particles in space where one particle moves between the others. In the first step, this code is to be extended to handle multiple free particles:

1. In reality, the planets/moons/..., i.e. every object in space, move as well, i.e. is subject of attraction from all other bodies. Extend the code such that all objects interact with each other and move.

2. Make the code fuse two bodies into one body if they collide with each other. Collision means the bodies are closer than 1e-8. The new body shall have a new mass that is equal to the sum of the masses of the original bodies and an averaged velocity (derive the formulae and present it in the report; you might want to implement some momentum-preserving scheme, i.e. mass-averaged result velocities, but the assignment is about the coding not about realistic physics, so a simple average of velocity components does the job).

3. he template code uses a fixed time step. Augment the code such that an appropriate time step size is chosen and no collisions are left out, i.e. bodies do not fly through each other.

Step 2 – numerical experiments
With the three functional extensions in place, conduct the following numerical studies:

- *Consistency/convergence:* Study two particles. Choose their velocity and initial position such that they directly collide with each other. Create a table where you document the resulting/merged particle's position. Study this position for various time step size choices (switching off the adaptive time step size choice from Step 1 question 3 above) and compare these positions to runs where you switch on the feature from Step 1 question 3. Can you uncover the convergence order of your scheme? Compare the obtained accuracy also to cost in terms of time step count.
- *Complexity:* Run your code for 10, 100, 1,000, 10,000 particles placed randomly in space. Derive the runtime complexity of the code and compare it to your experimental data.
- *Statistics:* Extend your code such that it keeps track of the number of bodies over time. Create a plot that shows how the total number of particles decreases over simulation time as particles merge.

Step 3 – a shared memory simulation
Parallelise the code with OpenMP and also ensure that the statistical evaluation from Step 2 is parallelised, too.

Step 4 – scaling experiments
Repeat the experiments from Step 2 to ensure that your modifications did not break the code. From hereon, create scaling plots for 10-10,000 particles. You are strongly encouraged to use a University machine for your plots that has at least 4 cores, i.e. you present a scaling plot than spans at least 1,2,3 and 4 cores. If you have a more powerful machine at home, you are free to use this machine. Clarify explicitly in your report the machine specifica.

Answer the following questions:

1. How does the scalability for very brief simulation runs depend on the total particle count?
2. Calibrate Gustafson's law to your setup and discuss the outcome. Take your considerations on the algorithm complexity into account.
3. How does the parallel efficiency change over time if you study a long-running simulation?

Step 5 – a distributed memory simulation
Design a strategy how to parallelise your code with MPI. No implementation is required, i.e. it is a gedankenexperiment.

Submission format
You have to submit exactly two files: a C/C++ source code file ending with .c or .cpp and one pdf file. Reports submitted as word file or other formats will not be accepted. The single source code file has to contain the realisation of all Steps 1-3. Keeping everything in one file is not good software engineering practice but simplifies marking. If more than one code file is submitted, no code will be taken into account and the rubrics will be marked with 0. While you may change the code, you may

not alter usage, i.e. it has to accept exactly the input arguments as the template on DUO. The report may not exceed two pages (incl. pictures/figures) for the Steps 2,4 and 5, i.e. it may not exceed six pages in total.

Feedback sheet

Criterion	Mark	Comments
All three features of Step 1 are implemented correctly.	(/ 40)	
All three questions from Step 2 are answered correctly.	(/ 40)	
Quality of presentation of simulation outputs for Step 2 (graphs, snapshots and notably videos).	(/ 20)	
The code is parallelised incl. the statistics on the total number of objects.	(/ 30)	
All three questions from Step 4 are answered correctly.	(/ 30)	
A clear concept for MPI parallelisation is presented.	(/ 20)	
Quality of writing and quality of presentation of simulation outputs (graphs, snapshots).	(/ 20)	

Total: (/ 100+100)

The first three rubrics fall into remit of the submodule Numerical Algorithms. The remaining rubrics belong to Parallel Programming.