Parallel Computing: Homework 4

Introduction

In this assignment, you will look at N-body simulation[1]. In simulation of dynamical systems, N-body simulation allows you to simulate how particles (on which forces, such as gravity, can be exerted) behave.

The provided code contains a sequential implementation of N-body simulation. It uses a simple approach that discretises the passage of time in 'time steps'. For each time step, the implementation will first compute the force each particle exerts on each other particle (pairwise), which requires n^2 computations. Then, the total force exerted on each particle will be computed (a simple sum), which will be used to then compute the movement in the current time step.

In addition to this programming assignment, there will once again be exercises on Nestor in the style of previous weeks. Make sure to not forget about finishing these!

Objective

The objective of this assignment is to parallelise the provided sequential implementation. If you examine the code, you might already spot some ways you can approach this - think carefully about what you parallelize, as this could impact the speed-up you can achieve. It should be possible to achieve around 1.5x speed-up on 2 cores with a simple approach, while a more elaborate implementation can get up to 2x speed-up.

You should also report on the efficiency of your parallel implementation: you will be provided with 2 input data sets that you need to put in your implementation and measure the runtime for different thread counts. This efficiency should be evaluated on the Peregrine computer cluster. Instructions on how to access Peregrine will be provided separately.

Execution

The code that is provided should be compiled using `gcc -O3 -Wall nBody.c -lm -lpthread`, after which you will be able to execute the code using `./a.out t n [input]`, where `t` is the number of time steps to compute, `n` is the number of threads to use, and `[input]` is the name of the input file to use. The output will consist of the final positions of the particles, as well as some debugging output including some time analysis.

Deliverables

We ask you to submit your code to Themis, which will only perform some basic sanity checks and will not extensively test the efficiency of your implementation. Themis will check that the output is still correct, ensure that the efficiency of serial execution is not hurt by the parallel implementation, and check whether you achieved at least *some* speed-up when running on 2 threads.

[1] `https://en.wikipedia.org/wiki/N-body_simulation`

In addition, we ask you to submit (on Themis) a *short* and **concise** report (PDF, no more than 2 pages) outlining your approach to the parallelisation. This report should contain the following table, completed with the performance results you obtained on Peregrine. The LaTeX code for this table can be found on Nestor. Please note the following:

- The execution time should be reported in seconds
- The speed-up and efficiency should be calculated relative to the execution time for 1 thread
- Note that there is a difference between the 'Serial' and '1' column: for 'Serial' we ask you to run the *original, unmodified* code as provided, and for the '1' column we ask you to run your modified, parallelised code with 1 thread. If there are siginificant differences between these two, please elaborate on that in your report.

Input file	Timesteps	Metric	Serial	1	2	4	8	16
`performanceTest1.in`	10 000	Execution time	X	X	X	X	X	X
		Speed-up	Y	Y	Y	Y	Y	Y
		Efficiency	Z	Z	Z	Z	Z	Z
`performanceTest2.in`	3	Execution time	X	X	X	X	X	X
		Speed-up	Y	Y	Y	Y	Y	Y
		Efficiency	Z	Z	Z	Z	Z	Z

Grading

We will grade on the criteria *correctness*, *efficiency*, and *code style*:

Correctness corresponds to passing test cases on Themis. If you did not pass all test cases, you might still get partial points. A manual code check might override the results from Themis, e.g. if you hardcoded answers or circumvented explicit assignment instructions.

Efficiency corresponds to the results you report in the performance table. We will perform a check of these performance numbers on Peregrine ourselves, so do not be tempted to alter them! Particularly efficient or sophisticated implementations might be awarded some bonus points.

Code style includes, but is not limited to: sufficient documentation, clear naming of variables, proper code structuring, proper code formatting, etc. *This is not an exhaustive list!*

Our grading will always be based on the *most recent* submission on Themis. Even though you might have submitted a 'better' version before, we will only look at the most recent one.

In addition to the points obtained for the points above, please do not forget to make the exercises posted on Nestor!