NOVA SCHOOL OF SCIENCE & TECHNOLOGY
COMPUTER SCIENCE DEPARTMENT

Concurrency and Parallelism 2022-23

Lab 05: OpenMP Implementation of the *N-body problem*

Hervé Paulino & Alex Davidson

April 2023

Abstract

In this lab class you will aim to write a concurrent implementation of the "N-body problem", as well as experimenting with various optimisations that could improve performance.

1 Introduction

The "N-body problem" aims to predict the individual motions of a number of particles, or celestial objects, that interact with each other according gravitational mechanics. Solving the problem is motivated by wanting to predict the movements of the Sun, Moon, planets and stars. You can read more about this problem here:

`https://en.wikipedia.org/wiki/N-body_problem`

This lab will aim to make you think not only about how to use OpenMP to parallelise a programme, but also about how the functionality of a programme can be changed to make it more amenable to OpenMP parallelisation.

2 Task: Parallel Implementation of N-body problem

2.1 Sequential version

Clone the given version of a sequential implementation of the N-body problem by following the link

`https://classroom.github.com/a/BC9ryvlw`

In order to compile it in the terminal, you will need to use `cmake`, you can see the repository README.md for instructions. The README also provides instructions on how to run the programme.

2.2 OpenMP version

The OpenMP version is unimplemented at the moment, see the following function in the `sec/openmp/par_nbody_all_pairs.cpp` file:

```
void par_nbody_all_pairs::calculate_forces()
```

Once you have compiled and experimented with the sequential version and checked that you can run it, you can this to write the OpenMP concurrent version by implementing this function. During development, you can test whether your code is correct using the *test suite* referred to in the repository README.

Please remember to use git appropriately. Learn to work with branches and make a branch when you start a new phase from the list above, and merge your branch to your main version when the development is finished and appropriately tested/validated.

3 Task: Performance Analysis

As you have done previously, you should create plots of your programme's performance across a number of runs, when altering the size of the initial board, and the number of generations. Furthermore, you can also use the tools explained previously such as `perf`, `gprof`, and `pprof` to produce profiles of your implementation. You can then use appropriate visualisations (such as Call Graphs and Flame Graphs) to display which parts of your programme are the most performance-intensive.

3.1 Key questions to answer

Once you have created your performance visualisations, you can answer the following questions.

1. You have the opportunity to try parallelising multiple loops in the programme — which configuration grants the most speed-up and why?
2. What is the impact of using the `-O3` flag in the speed-up?
3. Which functions take the longest to execute?

4 Task: Further optimisations

The programme could be improved further, consider the following two tasks.

1. You have an array of `particle` structs, each with their own `x` and `y` coordinates that are accessed across threads. Can you think of an alternative of way of structuring the arrays and the coordinates to avoid threads having to access the same structs at the same time? Implement your proposed version and compare performance with your original implementation.

2. The computational complexity of the naive algorithm (that you have implemented) is $O(N^2)$, where N is the number of particles. However, it has been shown (for example, in the Barnes-Hut simulation [1], and more recent implementations such as [2]) that the complexity of the algorithm can be reduced to $O(N \log N)$, or even $O(N)$. The work of Gangavarapu et al. gives multiple OpenMP implementations that convey this optimisation. Use their pseudocode to implement some of these optimisations in your code, and then check that performance increases by the desired complexity factor.

References

[1] J. E. Barnes and P. Hut. "A hierarchical O(n-log-n) force calculation algorithm". In: *Nature* 324 (1986), p. 446.

[2] Tushaar Gangavarapu et al. "Parallel OpenMP and CUDA Implementations of the N-Body Problem". In: *Computational Science and Its Applications – ICCSA 2019*. Ed. by Sanjay Misra et al. Cham: Springer International Publishing, 2019, pp. 193–208. ISBN: 978-3-030-24289-3.