**CPS3227**

Concurrency, HPC and Distributed Computing Assignment: n-body Simulator

Gabriel Lawrence Sammut   
184395M  
[gabriel.sammut.13@um.edu.mt](mailto:gabriel.sammut.13@um.edu.mt)

Academic Year 2018/2019

**Introduction**

This document entails the problem specification, and approach taken to implement a set of performance enhancements carried out on an N-Body simulator. Amongst the changes applied to the simulator, most prevalent include optimizations of a parallel and distributed nature, so as to ensure the generation of particle position predictions in reasonable time.

This document is split into several categories, opening up with a brief description of the overall design and implementation utilized to optimise the N-Body simulator. It is then followed up by results and timings of the benchmarking process, and an evaluation of the newly implemented concurrent logic.

**Task Description (N-Body Problem)**

The N-Body Problem consists of a number of scattered bodies, each denoted with the following attributes:

* Mass (M)
* Velocity (V)
* Position (P0,P1)

This particular rendition of the N-body simulator is limited to a two-dimensional representation. A set of four input files denoting each particle are provided, and any prevailing tests in this report are based on these inputs:

* input\_64.txt
* input\_1024.txt
* input\_4096.txt
* input\_16384.txt

Each file respectively contains a number of particles as denoted by their naming convention, and are stored in the following format:

|  |  |  |
| --- | --- | --- |
| **M** | **P0** | **P1** |
| 6.27803 | -368.462 | -499.992 |
| 3.5948 | 32.7673 | -41.3501 |
| … | … | … |

Table 1 - Rendition of Input File

Original N-Body simulator timings were ranked as follows, each generated on a single processor to simulate the sequential nature of the proposed task:

|  |  |  |
| --- | --- | --- |
| **Particle Count** | **Processors** | **Time (s)** |
| 64 | 1 | 2.54 |
| 1024 | 1 | 295.41 |
| 4096 | 1 | 3527.29 |
| 16384 | 1 | 55965.57 |

Table 2 - Sequential N-Body Timings

As can be appreciated, the task required to process N-Body simulations of high order particle magnitudes is time intensive, particularly as N increases.

**Problem Design**

In order to decrease the time required to execute the proposed N-Body simulation, the problem has been tackled using three methods:

1. A naïve shared memory implementation
2. A distributed memory implementation
3. A Hybrid memory implementation, using a combination of distributed and shared memory computing

The Naïve shared memory implementation involves usage of the OpenMP [1] standard, allowing the concurrent and parallel computation of calculating all position and velocity vectors at every iteration of the N-Body simulator. The maximum amount of concurrency was limited only by the infrastructure hardware, which for the sake of this experiment was limited to twelve processers.

The distributed implementation of the proposed problem was carried out using message passing techniques based on the Open MPI [2] standard.  
This approach allowed the computation of position and velocity vectors to be done concurrently on separate nodes, diverging and converging all processed work done under a single node (assumed to be given rank 0). The maximum amount of distributed nodes utilized by the experiment was set to four.

The third and final implementation involves a hybrid approach using a combination of shared and distributed memory techniques. The aim was to utilize the processing advantages of both prior techniques together, in order to achieve optimum timings for the N-Body simulation. Limited only by the given hardware infrastructure, the best run timings were achieved when running the N-Body simulation and generating particle bodies on four concurrent nodes, each allocated twelve processors to further parallelize in a shared environment.

Concrete and finalized timings can be found in the **Evaluation** section.

**Implementation**

The original N-Body simulator was supplied in native C, and was later enhanced and compiled using C++11. Certain parts of the original supplied code base was re-written to accustom eventual added OpenMP and Open MPI logic.

For the shared memory implementation, particular care was given to sharing and privatisation of variables, so as to avoid data race issues, whilst ensuring optimum performance at best by utilizing shared parallel variables whilst ensuring that the original sequential functionality is retained.

For the distributed memory implementation, particular use of the Open MPI collective methods [3] was used, including:

* MPI\_Scatter
* MPI\_Gather

Each make use of self-in-built code barriers, ensuring synchronization between all processing nodes at every iteration of the N-Body simulation.

**Results**

Each test was executed three times, and the average was recorded for each test.

**Evaluation**

**Conclusion**

**References**

[1] R. Friedman, R. Friedman, R. Friedman and R. Friedman, "Home - OpenMP", *OpenMP*, 2018. [Online]. Available: http://www.openmp.org/. [Accessed: 11- May- 2018].

[2] "Open MPI: Open Source High Performance Computing", *Open-mpi.org*, 2018. [Online]. Available: https://www.open-mpi.org/. [Accessed: 11- May- 2018].

[3]"A Comprehensive MPI Tutorial Resource · MPI Tutorial", *Mpitutorial.com*, 2018. [Online]. Available: http://mpitutorial.com/. [Accessed: 11- May- 2018].