CPS3227

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

Gabriel Lawrence Sammut
184395M
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 prediction of two dimensional particle coordinates in reasonable and acceptable time.

This report 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 Open MP [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 were re-written to accustom eventual added Open MP 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 result was recorded for each test. Timings were taken using Open MP's omp_get_wtime() function, and were scoped only for simulations of the N-Body problem, excluding any time overhead for simulation initialization and teardown.

This research has been carried out using computational facilities procured through the European Regional Development Fund, Project ERDF-080 'A Supercomputing Laboratory for the University of Malta' [5]

Open MP (x64)

CPU Count (n)	Average Time (s)
1	2.54
2	1.89
3	4.49
4	2.65
5	5.31
6	2.50
7	2.44
8	2.22
9	2.41
10	5.26
11	4.06
12	4.98

Table 3 - OPENMP (64 Particle Count)

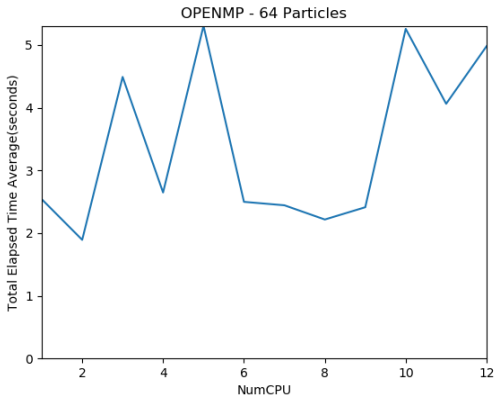

Figure 1 - OPENMP (64 Particle Count)

Open MP (x1024)

CPU Count (n)	Average Time (s)
1	295.41
2	125.22
3	81.67
4	75.89
5	61.89
6	47.78
7	54.01
8	44.75
9	51.85
10	59.23
11	40.23
12	28.38

Table 4 - OPENMP (1024 Particle Count)

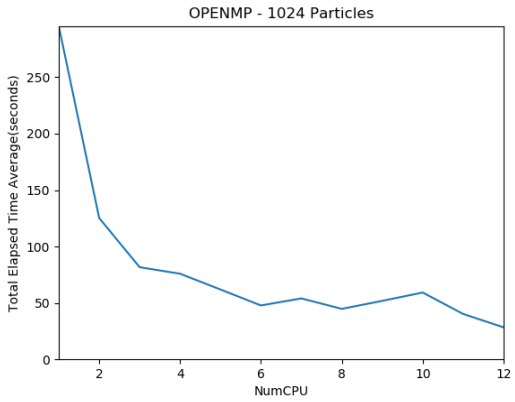

Figure 2 - OPENMP (1024 Particle Count)

Open MP (x4096)

CPU Count (n)	Average Time (s)
1	3527.29
2	1782.95
3	1235.58
4	902.58
5	739.72
6	620.00
7	536.37
8	467.00
9	420.01
10	380.92
11	348.55
12	321.23

Table 5 - OPENMP (4096 Particle Count)

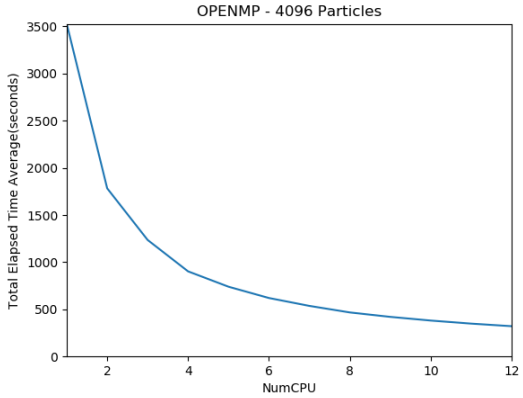

Figure 3 - OPENMP (4096 Particle Count)

Open MP (x16384)

CPU Count (n)	Average Time (s)
1	55965.57
2	28018.87
3	18696.80
4	14041.33
5	11255.93
6	9381.40
7	8068.15
8	7086.65
9	6286.76
10	5663.57
11	5148.37
12	4745.54

Table 6 - OPENMP (16384 Particle Count)

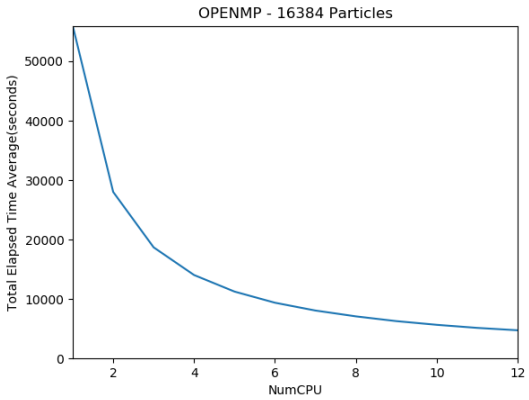

Figure 4 - OPENMP (16384 Particle Count)

Open MPI (x64)

Node Count (n)	Average Time (s)
1	2.83
2	2.33
3	2.39
4	3.45

Table 7 - OPENMPI (64 Particle Count)

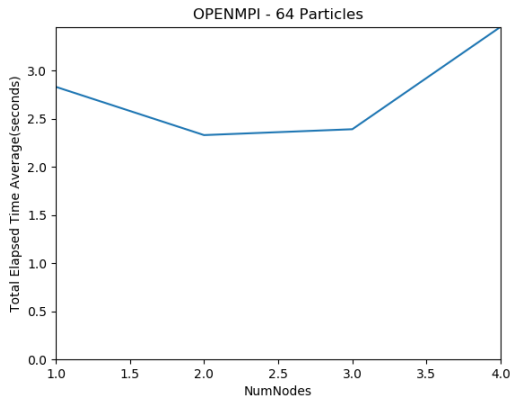

Figure 5 - OPENMPI (64 Particle Count)

Open MPI (x1024)

Node Count (n)	Average Time (s)
1	303.67
2	162.06
3	112.90
4	86.63

Table 8 - OPENMPI (1024 Particle Count)

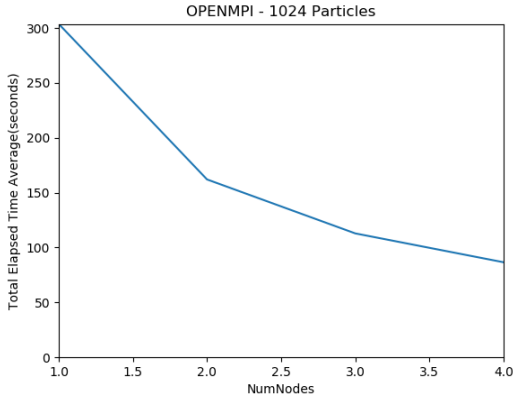

Figure 6 - OPENMPI (1024 Particle Count)

Open MPI (x4096)

Node Count (n)	Average Time (s)
1	3603.28
2	1835.38
3	1248.46
4	970.01

Table 9 - OPENMPI (4096 Particle Count)

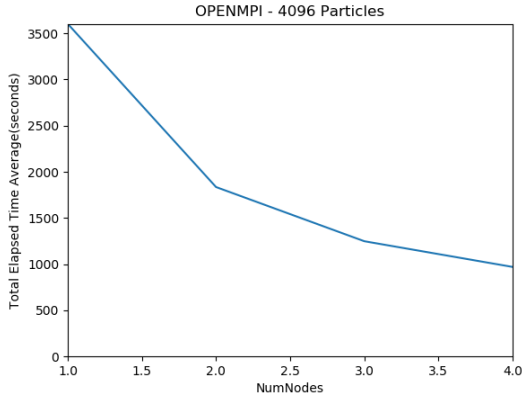

Figure 7 - OPENMPI (1024 Particle Count)

Open MPI (x16384)

Node Count (n)	Average Time (s)
1	56906.37
2	28214.8
3	18823.63
4	14022.17

Table 10 - OPENMPI (16384 Particle Count)

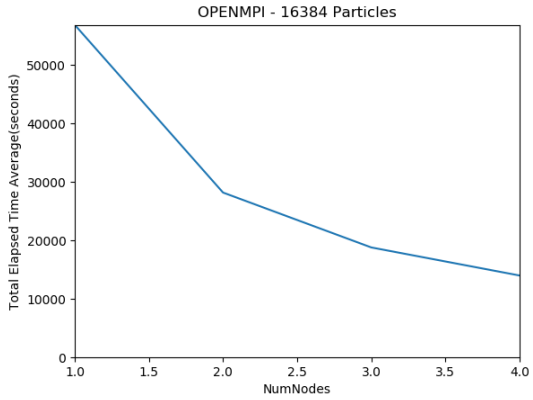

Figure 8 - OPENMPI (16384 Particle Count)

Open HYBRID – 1 Node (x64)

CPU Count (n)	Average Time (s)
1	4.37
2	3.93
3	3.70
4	1.90
5	2.05
6	1.80
7	5.58
8	4.35
9	5.12
10	4.47
11	3.78
12	2.05

Table 11 - OPEN HYRBRID(1) (64 Particle Count)

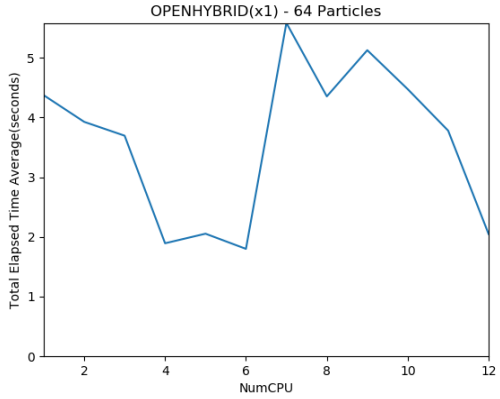

Figure 9 - OPEN HYBRID(1) (64 Particle Count)

Open HYBRID – 1 Node (x1024)

CPU Count (n)	Average Time (s)
1	292.26
2	126.65
3	92.03
4	78.72
5	75.64
6	76.31
7	104.81
8	69.05
9	83.06
10	71.68
11	51.85
12	46.45

Table 12 - OPEN HYBRID(1) (1024 Particle Count)

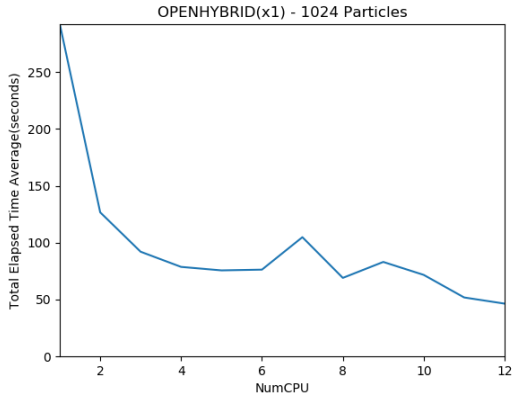

Figure 10 - OPEN HYBRID(1) (1024 Particle Count)

Open HYBRID – 1 Node (x4096)

CPU Count (n)	Average Time (s)
1	3526.58
2	1791.17
3	1199.87
4	909.24
5	734.71
6	618.03
7	879.76
8	773.79
9	716.90
10	694.47
11	666.93
12	637.17

Table 13 - OPEN HYBRID(1) (4096 Particle Count)

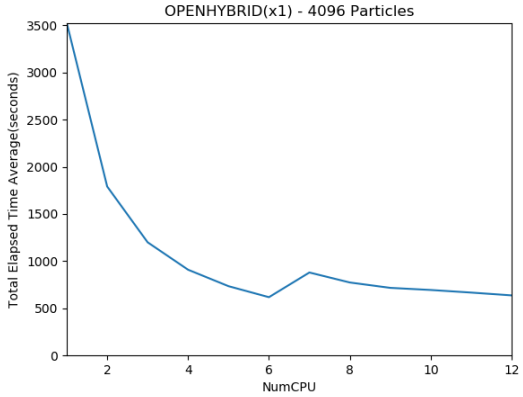

Figure 11 - OPEN HYBRID(1) (4096 Particle Count)

Open HYBRID – 1 Node (x16384)

CPU Count (n)	Average Time (s)
1	56066
2	28052.73
3	18675.6
4	14095.57
5	11209.37
6	9334.55
7	12441.57
8	11056.23
9	9879.42
10	10160.53
11	9899.44
12	9453.85

Table 14 - OPEN HYBRID(1) (16384 Particle Count)

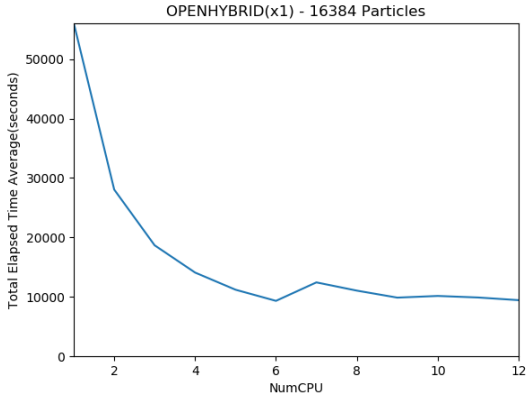

Figure 12 - OPEN HYBRID(1) (16384 Particle Count)

Open HYBRID – 2 Node (x64)

CPU Count (n)	Average Time (s)
1	3.06
2	9.21
3	4.67
4	11.66
5	3.65
6	3.87
7	4.69
8	4.64
9	2.43
10	2.44
11	2.29
12	1.99

Table 15 - OPEN HYBRID(2) (64 Particle Count)

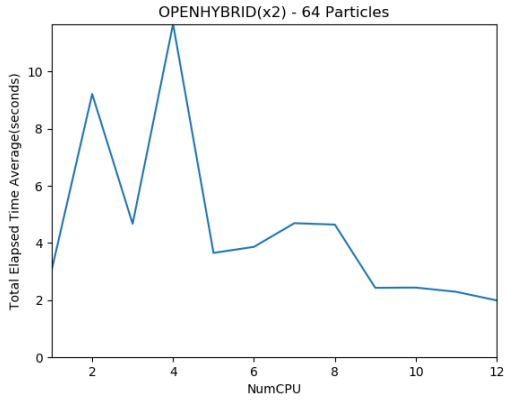

Figure 13 - OPEN HYBRID(2) (64 Particle Count)

Open HYBRID – 2 Node (x1024)

CPU Count (n)	Average Time (s)
1	158.91
2	127.09
3	69.47
4	72.05
5	60.34
6	38.08
7	61.48
8	60.36
9	49.19
10	46.81
11	47.25
12	59.16

Table 16 - OPEN HYBRID(2) (1024 Particle Count)

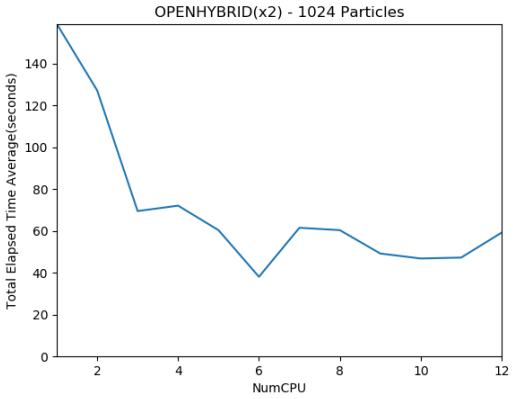

Figure 14 - OPEN HYBRID(2) (1024 Particle Count)

Open HYBRID – 2 Node (x4096)

CPU Count (n)	Average Time (s)
1	1806.21
2	1764.32
3	624.55
4	902.03
5	384.36
6	328.80
7	479.72
8	483.13
9	408.35
10	384.17
11	366.75
12	411.03

Table 17 - OPEN HYBRID(2) (4096 Particle Count)

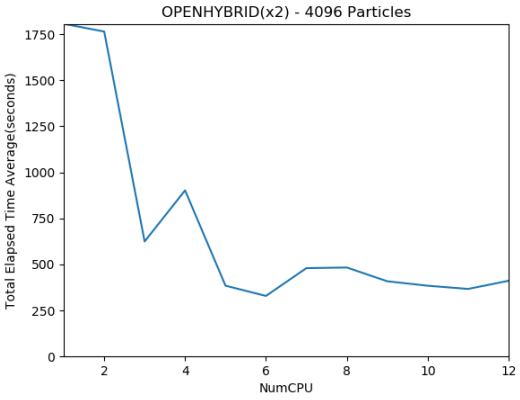

Figure 15 - OPEN HYBRID(2) (4096 Particle Count)

Open HYBRID – 2 Node (x16384)

CPU Count (n)	Average Time (s)
1	28347.37
2	27471.3
3	9512.597
4	12724.07
5	5807.523
6	4924.363
7	6569.91
8	6954.893
9	5473.77
10	5588.83
11	5227.747
12	5807.657

Table 18 - OPEN HYBRID(2) (16384 Particle Count)

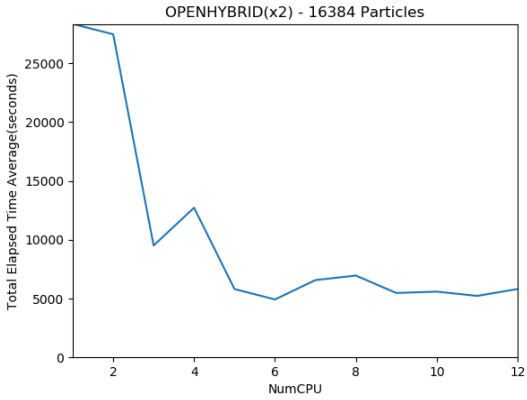

Figure 16 - OPEN HYBRID(2) (16384 Particle Count)

Open HYBRID – 3 Node (x64)

CPU Count (n)	Average Time (s)
1	3.56
2	3.73
3	12.27
4	1.90
5	12.31
6	17.35
7	5.92
8	4.28
9	3.06
10	5.41
11	6.87
12	2.49

Table 19 - OPEN HYBRID(3) (64 Particle Count)

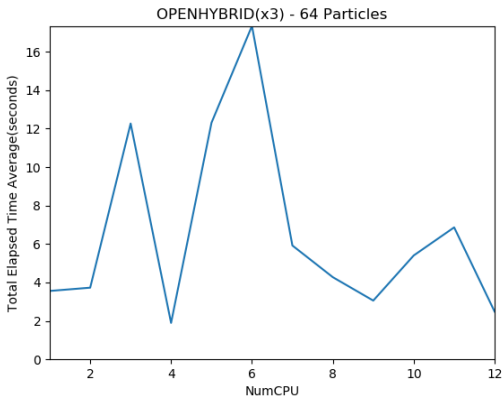

Figure 17 - OPEN HYBRID(3) (64 Particle Count)

Open HYBRID – 3 Node (x1024)

CPU Count (n)	Average Time (s)
1	113.81
2	59.72
3	82.52
4	39.89
5	72.47
6	71.44
7	58.51
8	39.72
9	44.07
10	39.28
11	39.15
12	25.09

Table 20 - OPEN HYBRID(3) (1024 Particle Count)

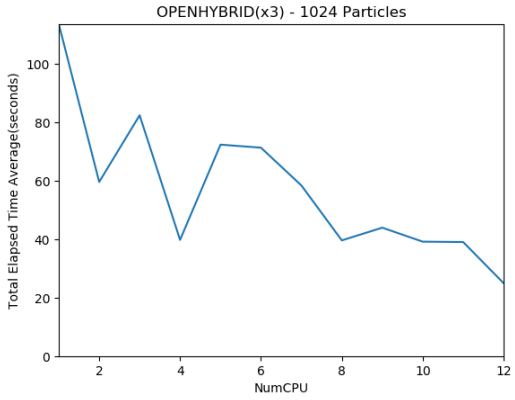

Figure 18 - OPEN HYBRID(3) (1024 Particle Count)

Open HYBRID – 3 Node (x4096)

CPU Count (n)	Average Time (s)
1	1216.12
2	621.04
3	795.65
4	325.44
5	463.97
6	392.25
7	361.44
8	313.11
9	320.77
10	301.67
11	296.80
12	236.91

Table 21 - OPEN HYBRID(3) (4096 Particle Count)

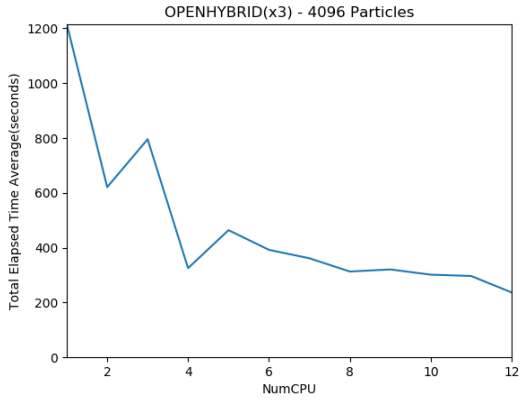

Figure 19 - OPEN HYBRID(3) (4096 Particle Count)

Open HYBRID – 3 Node (x16384)

CPU Count (n)	Average Time (s)
1	18699.80
2	9396.90
3	11667.87
4	4746.03
5	6780.74
6	5756.27
7	4969.26
8	4276.65
9	4463.79
10	4283.06
11	3902.33
12	3377.25

Table 22 - OPEN HYBRID(3) (16384 Particle Count)

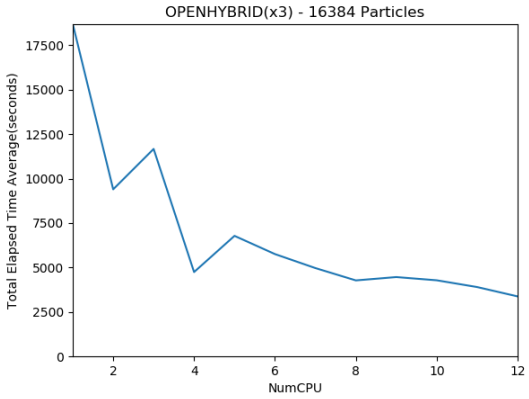

Figure 20 - OPEN HYBRID(3) (16384 Particle Count)

Open HYBRID – 4 Node (x64)

CPU Count (n)	Average Time (s)
1	4.08
2	4.15
3	2.39
4	1.86
5	2.20
6	3.57
7	1.84
8	3.64
9	3.63
10	4.72
11	4.91
12	3.12

Table 23 - OPEN HYBRID(4) (64 Particle Count)

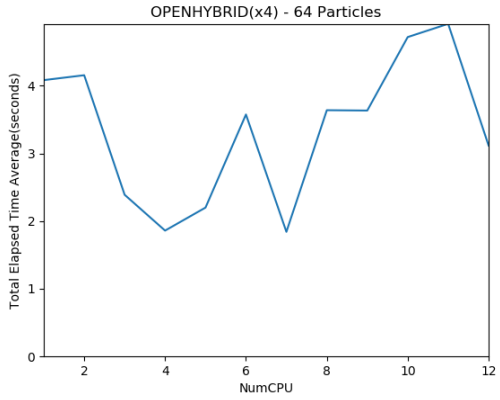

Figure 21 - OPEN HYBRID(4) (64 Particle Count)

Open HYBRID – 4 Node (x1024)

CPU Count (n)	Average Time (s)
1	90.46
2	51.14
3	41.90
4	30.89
5	30.83
6	43.43
7	38.61
8	32.24
9	31.81
10	30.13
11	26.14
12	29.63

Table 24 - OPEN HYBRID(4) (1024 Particle Count)

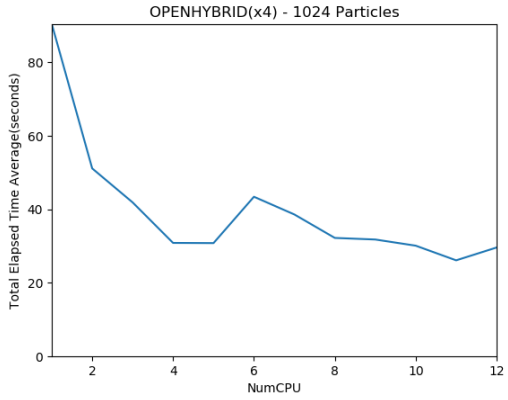

Figure 22 - OPEN HYBRID(4) (1024 Particle Count)

Open HYBRID – 4 Node (x4096)

CPU Count (n)	Average Time (s)
1	941.48
2	477.70
3	333.06
4	252.86
5	208.60
6	275.56
7	268.94
8	247.02
9	230.60
10	210.25
11	201.45
12	198.46

Table 25 - OPEN HYBRID(4) (4096 Particle Count)

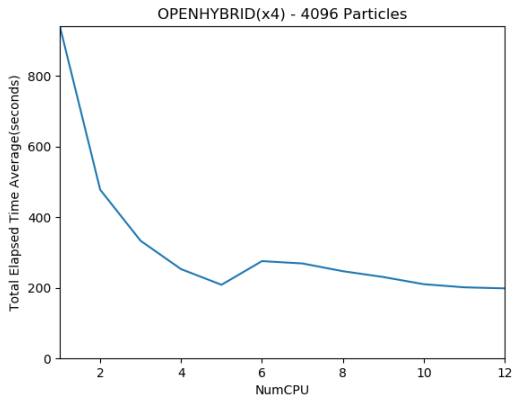

Figure 23 - OPEN HYBRID(4) (4096 Particle Count)

Open HYBRID – 4 Node (x16384)

CPU Count (n)	Average Time (s)
1	14246.03
2	7156.62
3	4811.52
4	3624.62
5	2923.59
6	3879.87
7	3662.89
8	3293.55
9	3446.42
10	2922.87
11	2685.54
12	2673.15

Table 26 - OPEN HYBRID(4) (16384 Particle Count)

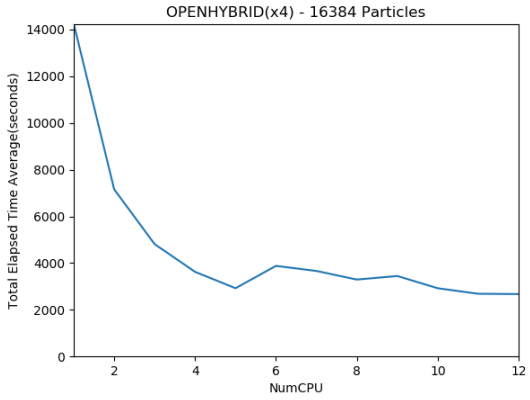

Figure 24 - OPEN HYBRID(4) (16384 Particle Count)

Evaluation

Several comments can be said with respect to the above results and visualizations:

1) Best observed result was using a hybrid approach (Open MP + Open MPI combined). Using a combination of four nodes, each running a portion of the problem concurrently using twelve threads, the best timing of 2673 seconds was achieved.
2) Result speed up seems to exhibit a relationship with the amount of growing nodes/threads upon which they were generated. The obtained curves (particularly so for runs using greater than 1024 particles) behaves in accordance to Gustafson's Law [4], as the distributed work performs much more elegantly on larger distribution of nodes/threads, with larger volumes to process.
3) For smaller runs (incorporating 64 particles), no particular correlation was established. This is presumed partially a cause of very little data to process with respect to the number of threads/processes invoked. This usually happens when the amount of data is too small to process in a parallel fashion, resulting in time overhead incurred by intra-process/intra-node communication.
4) Perhaps an anomaly in the extracted result timings, for runs using an odd number of threads, there were observed spikes in time computed in relation to runs when executed using an even number of concurrent threads. Initial assumptions and reasoning for such an anomaly can be explained as not distributing the amount of work equally between all processes, thus allocating more work on particular threads resulting in all other threads to wait until all threads have finished processing.
5) Most notable in the hybrid runs, time spikes when using a distribution of 5-7 threads per node were observed, which displayed different behaviour then the expected. This can be contributed to such runs being executed separately than the rest.

The following performance speed-ups were calculated and plotted, each based on the various input files. For the smaller particle files, the expected speed-up curve dips all most from the beginning, rendering any implementations of 2 or more nodes being less effective than a sequential version of the N-Body simulator.

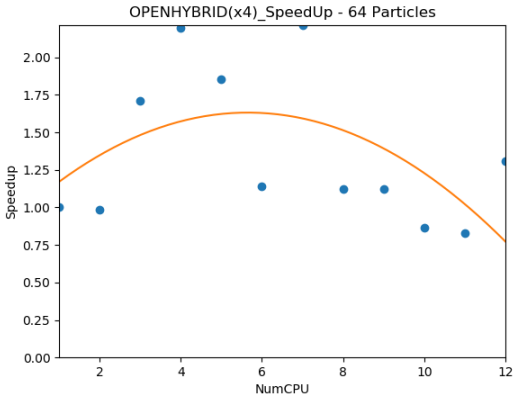

Figure 25 - OPEN HYBRID (64 Particle Speed-Up)

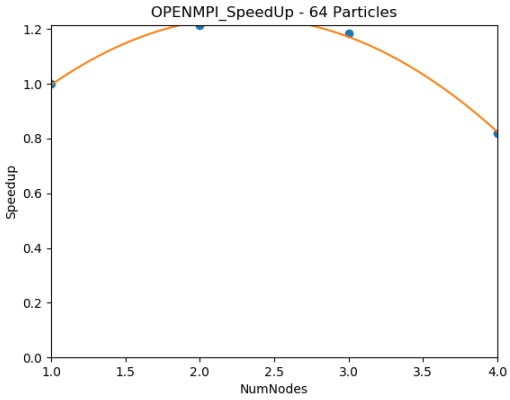

Figure 26 - OPEN MPI (64 Particle Speed-Up)

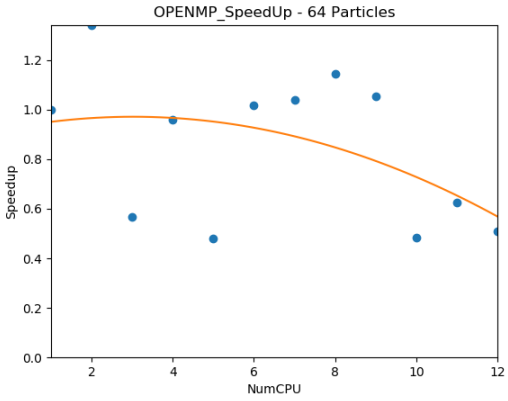

Figure 27 - OPEN MP (64 Particle Speed-Up)

These odd readings and plots behave in accordance to Amdahl's Law of Speedup [4]. This can be contributed mainly to the low number of particles and the high amount of inter-process synchronisation required to compute the N-Body simulation, effectively spending more time in organizing and coordinating processes together rather than the actual particle computations.

The 1024, 4096, 16384 particle runs were similar to each other in terms of speed up. Observed speed up behaved as expected for the Hybrid runs, hitting a drop in overall timing with the more processors added to the computation. Interestingly, runs for Open MPI and Open MP separately behaved in a linear fashion. This phenomenon could be a result of too little data points taken from the experiment, especially so in the case of the Open MPI runs. It could also mean that the threshold point at which overall processor efficiency starts to dip was yet to be established, suggesting that the N-Body simulation could have been parallelized further provided there was enough hardware to support it (for Open MP and Open MPI individually). Each respective plot can be identified below:

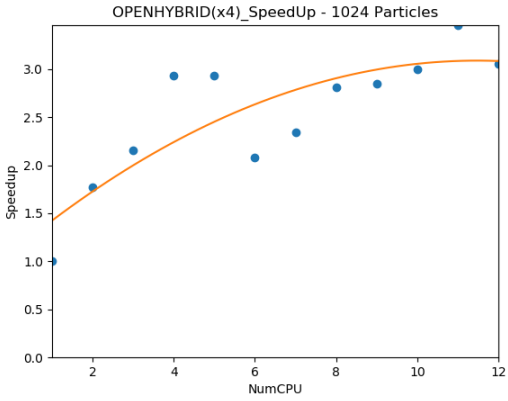

Figure 28 - OPEN HYBRID (1024 Particle Speed-Up)

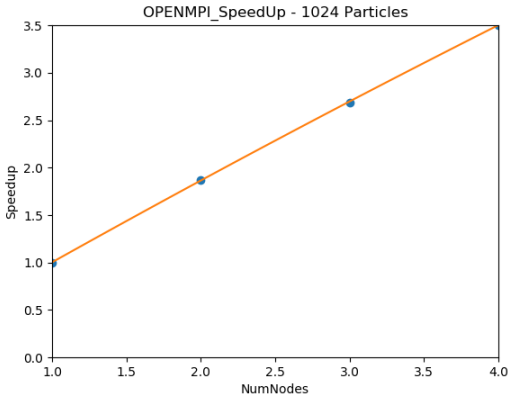

Figure 29 - OPEN MPI (1024 Particle Speed-Up)

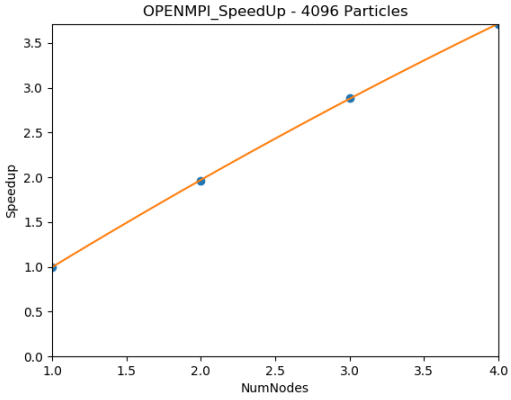

Figure 32 - OPEN MPI (4096 Speed-Up)

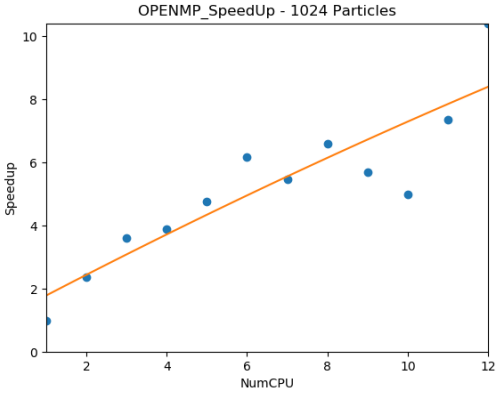

Figure 30 - OPEN MPI (1024 Particle Speed-Up)

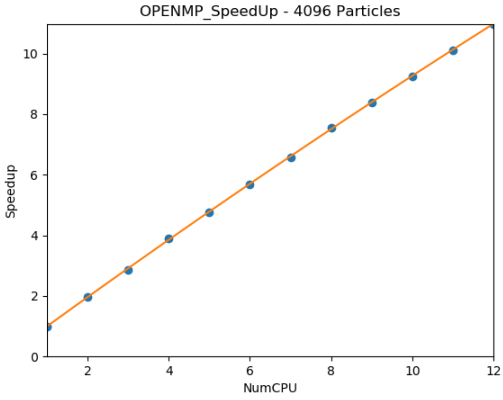

Figure 33 - OPEN MP (4096 Speed-Up)

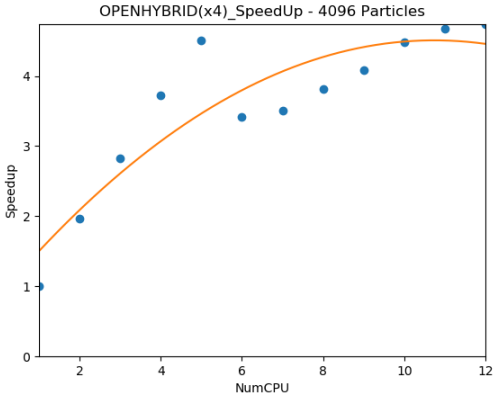

Figure 31 - OPEN HYBRID (4096 Speed-Up)

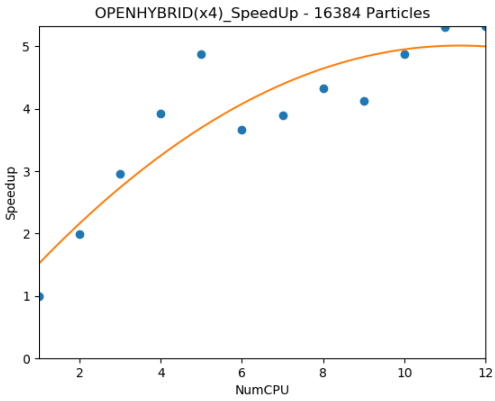

Figure 34 - OPEN HYBRID (16384 Speed-Up)

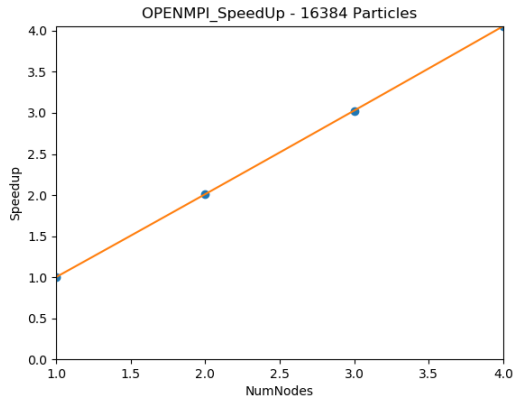

Figure 35 - OPEN MPI (16384 Speed-Up)

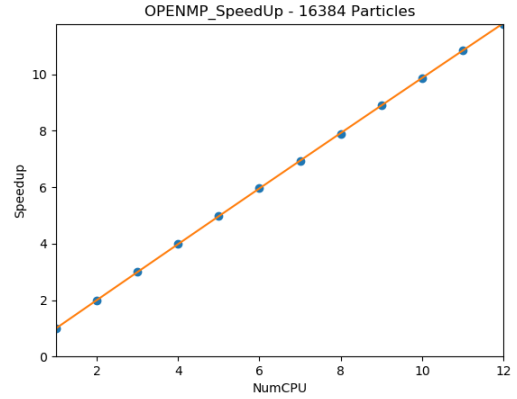

Figure 36 - OPEN MP (16384 Speed-Up)

Conclusion

Although several anomalies can be observed from the extracted timings, the overall results seem in accordance to the expected behaviour of node speedup in relation to processed particle runs. The best run was obtained by splitting the problem to run concurrently on a single machine, and then going further to distribute any work done on separate nodes, each running a portion of the problem concurrently, and synchronizing under a master node at every iteration of the problem. For the largest file (containing 16384 particles), original timings were timed to take almost 16 hours (15 hours, 54 mins) to compute. After all performance enhancements were added to the N-Body simulator, the same run was timed to take approximately 45 minutes, resulting in a timely gain of more than 15 hours.

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]

[4]J. Gustafson, "Reevaluating Amdahl's law", *Communications of the ACM*, vol. 31, no. 5, pp. 532-533, 1988..

[5]"overview_of_albert [Grid Wiki]", *Secure.um.edu.mt*, 2018. [Online]. Available: https://secure.um.edu.mt/itservices/gridwiki/doku.php/overview_of_albert. [Accessed: 17- May- 2018].

Appendix

Complete work can be viewed from either of both links:

- https://github.com/eldrad294/CPS3227
- https://drive.google.com/open?id=1nHMWfXJ4dbdx9Au2jK9ayE_3KoxYvzkN