A Distributed-Memory Simulation of the Andromeda-Milky Way Collision

ISITHA SUBASINGHE, University of Melbourne, Australia

In approximately 5 billion years [14], our Milky Way galaxy and the distant Andromeda galaxy will be merged into the Andromeda-Milky Way galaxy. We explore how a simulation of this collision could be realised through the usage of OpenMPI[1] on the Spartan HPC cloud chimera [12] . The report explores how the Barnes-Hut [1] algorithm used for solving general n-body problems such as planetary body simulations could be trivially parallelised using distributed processing of the galaxies on an OpenMPI cluster. Our implementation of the Barnes-Hut algorithm over OpenMPI was able to offer a near linear speedup with respect to the number of processes used and offered a 41x speedup with 64 cores.

CCS Concepts: • **Computing methodologies → Parallel algorithms**.

1 BACKGROUND

The problem of solving general n-body problems is quite simple in some sense and has arguably been known for hundreds of years following Isaac Newtons formalisation of the law of universal gravitation despite not being able to solve n-body problems in terms of integrals for n ≥ 3. The force exerted on an object j by i can be given through:

$$\mathbf{F}_{j,i} = -G\frac{m_i m_j}{\left|\mathbf{r}_{ji}\right|^2}\hat{\mathbf{r}}_{ji} \tag{1}$$

where:

$$\mathbf{r}_{ji} = \left|\mathbf{r}_j - \mathbf{r}_i\right|$$

$$G \text{ is the gravitational constant}$$

$$\hat{\mathbf{r}}_{ji} = \frac{\mathbf{r}_j - \mathbf{r}_i}{\left|\mathbf{r}_j - \mathbf{r}_i\right|}$$

Following this understanding of basic gravitational mechanics it becomes trivial to devise an exact computational technique for solving the planetary body simulation problem. Continuing from (1), we can derive the force a set of bodies may have on object j by:

$$\mathbf{F_j} = \sum_{\substack{i=0 \\ i\neq j}}^{n} -G\frac{m_i m_j}{\left|\mathbf{r}_{ji}\right|^2}\hat{\mathbf{r}}_{ji} \tag{2}$$

And from (2) we can finally derive the equation for updating the velocity and position of a particular body through:

$$\begin{aligned} \mathbf{v_j}' &= \mathbf{v_j} + \frac{\mathbf{F_j}}{m_j} * \Delta t \\ \mathbf{r_j}' &= \mathbf{r_j} + \mathbf{v_j} * \Delta t \end{aligned} \tag{3}$$

We can note that the runtime complexity of solving the planetary body simulation problem through (3) is $O(n^2)$ due to the need to compare every body with every other body in the system. With the large number of planetary bodies in the Milky-Way alone, it becomes obvious that this naive implementation is inadequate for the task of simulating the Andromeda-Milky Way collision. It is important to note that when simulating planetary bodies of this size over such a large time-span, the act of performing the simulation accurately and in a timely manner becomes an infeasible problem, therefore making sacrifices in accuracy for increased runtime performance is an acceptable outcome. The Barnes-Hut [1] algorithm introduced over 35 years ago, is among one of the most commonly used algorithms for solving general n-body problems such as planetary body simulations. The algorithm offers good performance characteristics with respect to the number of bodies it simulates, however its core advantage is the relative ease of parallelising the algorithm, these properties makes the Barnes-Hut algorithm ideal for our simulation of the Andromeda-Milky Way collision.

2 RELATED WORK

The Fast Multipole Method [6] was initially proposed to solve Gauss transforms in a performant yet accurate way but it still is capable of solving general n-body problems. The core advantage is the linear runtime $O(n)$ or more accurately $O(n * log(1/\epsilon))$ where ϵ is the error tolerance. As a result of the mentioned time complexity, the Fast Multipole Method benefits when there is a large number of bodies to be simulated, however it is not obvious if the real performance characteristics favour FMM and unfortunately there is no mechanical process to determine what values of n favour the Fast Multipole Method over other algorithms. Blelloch [2] explored the various differences in the algorithms for solving the general n-body problem, they observed that the Barnes-Hut algorithm can be significantly faster when the required accuracy required is low, even for extremely large planetary body simulations.

There have been various attempts to use the high levels of parallelism introduced in modern GPUs, some of them have been to simply perform an $O(n^2)$ “all-pairs” calculation such as in [13], quite surprisingly Nyland [13] was still able to offer a 50x speedup over the serial algorithm. Other researchers have explored as to how the Barnes-Hut algorithm could be implemented on a CUDA enabled GPU [3]. Interestingly Burtscher’s algorithm [3] was able to offer a 74x speedup with a 240 core GPU.

A unique, specialised approach to solving the n-body problem was proposed initially by Ito called GRAvity-PipE (GRAPE-1) [8], this approach was the development of a specialised computer (a co-processor) to perform the force calculations involved in the n-body problem, the result of using hardwired logic for the problem resulted in significant speedups. Following this initial research, there have been numerous and continuing improvements on this approach as shown by [9–11].

2.1 Parallel Algorithm

The proposal for this idea derives from the fact that the tree building part of the Barnes-Hut algorithm is relatively simple and can be done in $O(n)$ time, which is significantly faster than the $O(n * log(n))$ runtime of the computation step. Initial performance testing in the development environment demonstrated that the construction step

[1]See https://www.open-mpi.org/

only took about 0.2% of total computation time in the worst case. The algorithm was quite a simple variation on the sequential algorithm, we simply limited the process from calculating the force for id numbers that were not in its assigned processing range. A simple 'MPI_Allgatherv' instruction was used to broadcast the updated velocities and positions. This load balancing was quite simple to perform yet it demonstrated exceptional performance characteristics.

The parallel algorithm implemented did not make use of thread level parallelism via OpenMP [2] and instead opted for process level parallelism through OpenMPI. The justification for this is simply as a result of our initial testing that revealed that OctTree construction time dwarfs the runtime of the actual computation time. This made it obvious that a single threaded yet multi process implementation was a viable option, especially since we could avoid NUMA and cache related performance penalties.

2.1.1 Integration Method. The integration method used to solve the differential equation may have an significant effect on the accuracy of the simulation. Traditional algorithms used such as the Runge-Kutta method suffer from unbounded error. As the number of iterations of the simulation increase, it is possible that the error grows large enough such that the final result would not be an accurate simulation, this is especially important when simulating such a large time span. In order to solve this a symplectic integration technique was used called the Leapfrog method [7]. Leapfrog is symplectic, meaning that it is time reversible, this is an important characteristic as it means that this technique conserves the energy in the system to some bounded error.

2.1.2 OctTree depth. OctTree depth has a significant impact on the runtime characteristics of the Barnes-Hut algorithm. This is ultimately as a result of how Barnes-Hut determines if an octant is worth computing exactly or not. From the algorithm for Barnes-Hut, note how θ (s/d) is calculated and checked to determine if an approximate centre of mass calculation should be used instead of an exact solution. Consider the case where we have a small maximum depth (d_s) to our OctTree, this means on average $s_{d_s} > s_{d_l}$ where d_l indicates a larger OctTree depth, from this it is trivial to realise that $\theta_s > \theta_l$. This is an important recognition as having a smaller θ on average means that less octants will be computed exactly, leading to better performance characteristics. One more impact of the depth that we had to take into account for was the number of planets in a leaf octant, it is better for locality of reference that we have some contiguous data that we can trivially iterate over. Through experimental evaluation and taking the above considerations into account, an OctTree depth of 6 was established for the data used.

2.1.3 Theta Tolerance. The tolerance level for the θ value also has a significant impact on the runtime performance of the algorithm. Consider the case where the maximum θ tolerance is capped at 0, the algorithm then simply degrades into an overly complicated $O(n^2)$ "all-pairs" calculation. The θ value was determined through evaluation of papers such as [15] which showed that a θ value of 0.7-1.0 provided the best trade-off in terms of error and runtime.

[2]See https://www.openmp.org

2.1.4 Compiler optimisations. Compiler optimisations were heavily used to further increase the performance of the algorithm and frequent profiling of the program was done via the linux utility "perf" [3]. The standard library functions proved to be expensive in tight loops as a result of symbol lookup, static linking of the C++ standard library was performed in order to address this issue. Additional optimisation recommendations from [5] were evaluated and used where appropriate. A critical optimisation used was the "-march=native" flag passed onto the GNU compiler, this flag targets the exact architecture that we are compiling the program for, this results in further architecture level optimisations being possible such as usage of AVX-256.

3 EXPERIMENTS

3.1 Experiment Environment

Two experimentation environments were used throughout the development process. The development environment was used as a quick way to test and filter out ideas that weren't viable, while valid ideas were tested on the Spartan High Performance Computing cloud [12].

3.1.1 Development Environment.

- CPU - AMD Ryzen™ 7 3700X @ 3.6-4.4GHz
- Memory - 32GB
- Compiler - GNU 11.1.0

3.1.2 Spartan Environment.

- CPU - Intel(R) Xeon(R) CPU E5-2698 v3 @ 2.30-3.6GHz
- Memory Per Processor - 2GB
- Compiler - GNU 8.3.0

3.2 Results

The results were determined through running the Barnes-Hut algorithm for 30 iterations on the model B of Dubinski's work [4] and shown below. The data does not contain every star system in the two galaxies, in fact it is a much smaller representation at just 81,920 bodies. It is important to note that this simulation starts about 1 billion years from the collision, this is largely as a result of the initial 3-4 billion years being mostly of no consequence to us.

Core(s)	θ = 0.7, Time(s)	θ = 1.0, Time(s)
8	993.142s	884.421s
16	500.831s	446.234s
24	347.724s	318.101s
32	259.639s	237.971s
64	142.153s	120.661s

Table 1. Wall time per cores and θ value

[3]See https://perf.wiki.kernel.org/index.php/Main_Page

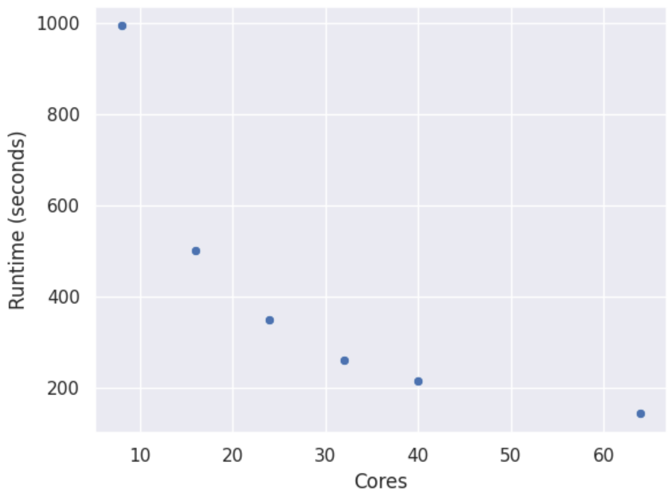

Fig. 1. Results observed $\theta = 0.7$

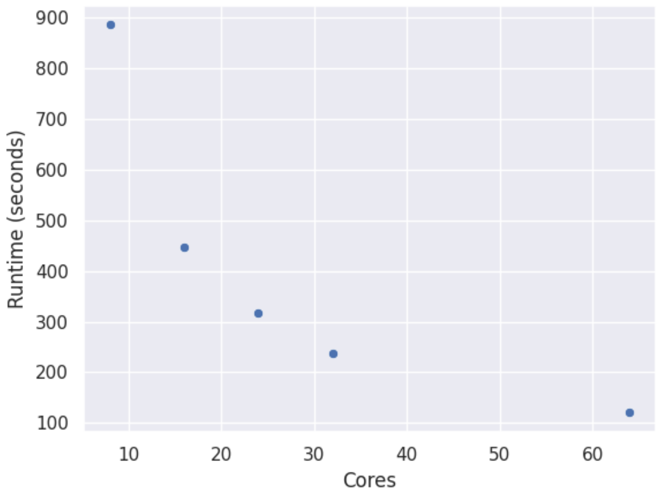

Fig. 2. Results observed $\theta = 1.0$

3.3 Discussion

Note: The sequential implementation was measured to have taken 5853.931s.

Through section 3.2 we can see that the runtimes for the algorithm are quite similar with the lower and upper bounds for our acceptable θ tolerance values that we determined earlier and therefore it makes sense to perform the full simulation with $\theta = 0.7$ as it will lead to a more accurate solution despite it being slightly slower.

An obvious limitation of our approach is the lack of accounting for the black holes at the centre of the two galaxies. When the simulation nears completion it is likely that the two black holes at the centre of both these galaxies will have a non trivial implication on how the simulation is played out. Unfortunately, no attempt was made on accounting for these gravitational effects as a result of its reliance on Einstein's theory of general relativity, while relativity can be simulated, it is much more difficult to do so than simple Newtonian mechanics.

Unfortunately, it is likely that the 81,920 bodies approaches the largest problem size that we can simulate using this method in a timely manner, further work is required to determine how more bodies could be simulated.

4 FURTHER WORK

4.1 Simulating General Relativity

As mentioned previously it could be beneficial to simulate Relativity instead of simple Newtonian based gravitation, solely as a result of the impacts of strong gravitation caused by the two black holes at the centre of both the Milky Way and the Andromeda galaxy.

5 CONCLUSION

We have demonstrated how OpenMPI may be used to simulate the gravitational mechanics of large n-body problems in an effective way. As shown previously we were able to gain significant reductions in runtime performance through our distributed processing strategy. The proposed algorithm was able to offer a 41x speedup with just 64 cores, reducing a runtime of nearly 100 minutes to about 2.5 minutes, making the timely simulation of Andromeda-Milky Way collision a feasible problem.

REFERENCES

[1] Josh Barnes and Piet Hut. 1986. A hierarchical O (N log N) force-calculation algorithm. *nature* 324, 6096 (1986), 446–449.
[2] Guy Blelloch and Girija Narlikar. 1997. A practical comparison of TV-body algorithms. *Parallel Algorithms: Third DIMACS Implementation Challenge* 30 (1997), 81.
[3] Martin Burtscher and Keshav Pingali. 2011. An efficient CUDA implementation of the tree-based barnes hut n-body algorithm. In *GPU computing Gems Emerald edition*. Elsevier, 75–92.
[4] John Dubinski, J Christopher Mihos, and Lars Hernquist. 1995. Using tidal tails to probe dark matter halos. *arXiv preprint astro-ph/9509010* (1995).
[5] Agner Fog. 2020. Optimizing software in C++.
[6] Leslie Greengard and John Strain. 1991. The fast Gauss transform. *SIAM J. Sci. Statist. Comput.* 12, 1 (1991), 79–94.
[7] Arieh Iserles. 1986. Generalized leapfrog methods. *IMA J. Numer. Anal.* 6, 4 (1986), 381–392.
[8] Tomoyoshi Ito, Junichiro Makino, Toshikazu Ebisuzaki, and Daiichiro Sugimoto. 1990. A special-purpose N-body machine GRAPE-1. *Computer Physics Communications* 60, 2 (1990), 187–194.
[9] Atsushi Kawai, Toshiyuki Fukushige, Junichiro Makino, and Makoto Taiji. 2000. Grape-5: A special-purpose computer for n-body simulations. *Publications of the Astronomical Society of Japan* 52, 4 (2000), 659–676.
[10] Junichiro Makino and Hiroshi Daisaka. 2012. GRAPE-8–An accelerator for gravitational N-body simulation with 20.5 Gflops/W performance. In *SC'12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–10.
[11] Junichiro Makino, Makoto Taiji, Toshikazu Ebisuzaki, and Daiichiro Sugimoto. 1997. Grape-4: A massively parallel special-purpose computer for collisional n-body simulations. *The Astrophysical Journal* 480, 1 (1997), 432.
[12] BERNARD MEADE, LEV LAFAYETTE, Greg Sauter, and DANIEL TOSELLO. 2017. Spartan HPC-Cloud Hybrid: Delivering Performance and Flexibility. https://doi.org/10.4225/49/58ead90dceaaa
[13] Lars Nylons, Mark Harris, and Jan Prins. 2007. Fast n-body simulation with cuda.
[14] Sangmo Tony Sohn, Jay Anderson, and Roeland P Van der Marel. 2012. The M31 velocity vector. I. Hubble Space Telescope proper-motion measurements. *The Astrophysical Journal* 753, 1 (2012), 7.

[15] Laurens Van Der Maaten. 2013. Barnes-hut-sne. *arXiv preprint arXiv:1301.3342* (2013).

A APPENDIX

Algorithm 1 Computes the new velocities and positions after one iteration of Barnes-Hut

```
1: procedure RunIteration
2:     bodies ⟵ planets in the system
3:     root ⟵ OctTree root
4:     rank ⟵ rank of OpenMPI process
5:     chunkSize ⟵ size of chunks
6:     size ⟵ number of processes
7:     n ⟵ placeholder variable for number of planets
8:     lowerBound ⟵ chunkSize*rank
9:     upperBound ⟵ chunkSize*(rank+1)
10:    newBodies ⟵ array of new bodies generated after simulation
11:    for i ⟵ $[0 \ldots n)$ do
12:        if bodies[i].id ≥ lowerBound && bodies[i].id < upperBound then
13:            newBody ⟵ simulateTimestep(root, bodies[i])
14:            newBodies.push(newBody)
       return newBodies
```

Algorithm 2 Distributed processing of the Barnes-Hut algorithm

```
1: procedure RunDistributedBhut
2:     planets ⟵ memory of where planets are to be held
3:     rank ⟵ rank of OpenMPI process
4:     size ⟵ number of processes
5:     n ⟵ placeholder variable for number of planets
6:     iters ⟵ number of iterations to run
7:     if $rank == 0$ then
8:         planets ⟵ ReadPlanets()
9:         n ⟵ length(planets)
10:    Broadcast(n)
11:    Broadcast(planets)
12:    $chunkSize \longleftarrow (n/size) + (n\%size)$
13:    for i ⟵ $[0 \ldots iters)$ do
14:        (origin, bounds) ⟵ OctTreeInitParams(planets)
15:        root ⟵ NewOctTree(planets, origin, bounds)
16:        localPlanets ⟵ RunIteration(planets, roots, chunkSize, rank, size)
17:        Allgather(localPlanets, planets)
```