Distributing Direct N-Body Simulations

Steven Johnson
Department of Computer Science and Software Engineering
California Polytechnic State University
San Luis Obispo, CA USA
sjohns98@calpoly.edu

Daniel Sabsay
Department of Computer Science and Software Engineering
California Polytechnic State University
San Luis Obispo, CA USA
dsabsay@calpoly.edu

***Abstract*—N-body simulations have been used by scientists to study solar systems and other celestial systems. These simulations, while they provide illuminating insight about the workings of our universe and the interaction of celestial objects, are computationally intensive. For large numbers of bodies, the required calculations can become prohibitively time consuming. In this paper, we aim to distribute an N-body simulation over multiple compute nodes, thus significantly improving the speed of calculation. We will use Cowell's method because it is 1) accurate, and 2) easy to implement. Cowell's method involves, for each celestial body b, summing the effects of all other bodies to calculate the total acceleration of b. The resulting acceleration vector is then integrated twice, once to compute the velocity vector, and again to compute the final position of b after a given time interval. Our implementation uses the MPI (Message Passing Interface) library for C and was tested on a single, quad-core machine. We compare the performance of two distributed solutions to the performance of the same simulation computed sequentially on a single node. The results we found indicate that computation time can be reduced significantly with a distributed (MPI) solution. On the other hand, our multithreaded MPI solution appeared not to improve performance.**

Keywords—n-body simulations, distributed computing, message-passing

I. Introduction

N-body simulations are used extensively in physics and astronomy to study the orbits of nearby celestial objects within our solar system, and to model the formation and evolution of star clusters, galactic phenomenon, and other systems in the universe. N-body simulations, and sometimes the more restricted 2-body and 3-body variants, are also used to model satellite orbits and other spacecraft trajectories. Various methods exist, from accurate but computationally intensive direct force summations (i.e. direct N-body, also known as Cowell's method), to more approximate methods such as the Barnes-Hut tree simulation. One must trade accuracy for performance.

A current unsolved problem involving N-body simulations is simulating entire galaxies, which according to [1] would require at least a few PFLOP/s of computational power, something not attainable with current technology on a single supercomputer. Thus, in order to further scientific research in this area, faster and more efficient simulations are required.

To enable faster execution of simulations, grid enabled distributed solutions are examined in [1] by creating an N-body simulation using the MPI-CH2 library, an MPI library allowing for grid-enabled distributed systems. Two different algorithms are employed: the copy algorithm (which maintains a tree of all bodies on each node, thus reducing communication required), and the memory reducing ring algorithm (which only maintains on each node a partial tree that the node will use to calculate the partial force integration between its local subset, after which it will send updated information to its neighboring node). These simulations were run on a set of four GRAPE supercomputers, along with an individual GRAPE computer, stationed in geographically separated parts of the world (one in Amsterdam, one in Pennsylvania, and two in Tokyo). The results were analyzed to determine the most effective algorithm and the point at which communication time becomes less of a factor than force calculations when measuring total execution time, with the aim of determining the viability of using a distributed grid of GRAPE computers for N-body simulations. The copy algorithm performed better than the ring algorithm in this environment because it minimized communication. However, for simulations with less than 10^4 bodies, communication time still accounted for a greater portion of execution time than force calculations. They also found that a single GRAPE computer would be more effective than a grid in simulations containing less than 3×10^6 bodies.

Using similar techniques to the ring algorithm in [1], in [2], researchers create a parallel N-body simulation but with distributed memory and message passing in order to determine if, as described in other research, it is inferior to shared memory in this application. They represent bodies using the Barnes-Hut algorithm with trees maintained on each node. Clusters of bodies were approximated into single nodes on the

tree which are then unpacked and calculated at a finer level by processors, thus reducing the time needed to span the tree. Forces are calculated similarly to the ring algorithm in that processors are only responsible for bodies within a certain area, and local trees are dynamically adjusted based on each processors' assigned area while only one global tree is maintained. Based on these improvements to the algorithm, there was no found slowdown as a result of the message passing overhead when using distributed memory. These results, while run on a parallel architecture, are still useful when considering our own implementation, as the memory is distributed and message passing is still used..

In [3], an N-body simulation is run on a network of three globally dispersed supercomputers with communication overhead accounting for only ~10% of computation time. From these experiments it can be seen that distributed N-body simulations across supercomputers are both technically feasible and advantageous. Unlike [3], our approach does not involve supercomputers, nor do we attempt to distribute across a global network, but we have taken inspiration from the approach of distributing work while minimizing communication overhead. Particles are distributed between the supercomputers using a 1-dimensional slab decomposition, which means that only 1-dimensional information needs to be transmitted to negotiate particle ownership. This decomposition also works well in a ring network topology where each supercomputer is connected to only two others.

In [4], a hybrid approach is implemented using HPX to distribute work among nodes and parallelize tasks within each node. Computations are distributed and parallelized across both space and time. Their implementation uses the Barnes-Hut algorithm, in which space is partitioned into regions (a.k.a "cubes") and all particles are assigned to a cube. As particles move over time, they may cross the boundaries of their initial cube, in effect leaving that cube. Instead of broadcasting that particle's information to all neighboring localities, it is sent to only one neighbor calculated from the particle's new position. Unique IDs are assigned to each locality so that one locality may deduce the ID of the destination in order to send the particle's information. If the particle does not belong to the first receiving locality, it will pass the particle to the next locality. In addition to this parallelization across space, their implementation allows for localities to begin computation on a future timestep even if not all necessary information from neighboring localities has been received. This reduces the impact of synchronizing all localities at every timestep. While this approach takes advantage of both distributed and parallel techniques, our implementation will focus solely on a distributed system using MPI.

While [1], [2], [3], [4] and much other work in this field employs various approximation methods to compute simulations with a large number of bodies, and while many of these approaches also employ hybrid parallelization schemes that utilize message passing between distributed nodes and parallelization within nodes, our work aims to improve performance of direct N-body simulations using a distributed system. We aim to determine under what simulation parameters, if any, a direct simulation can be accelerated via distributed computing techniques, utilizing distributed memory and message passing.

II. Implementation

We compared two different distributed algorithms for direct n-body simulations against a non-distributed, single-threaded implementation.

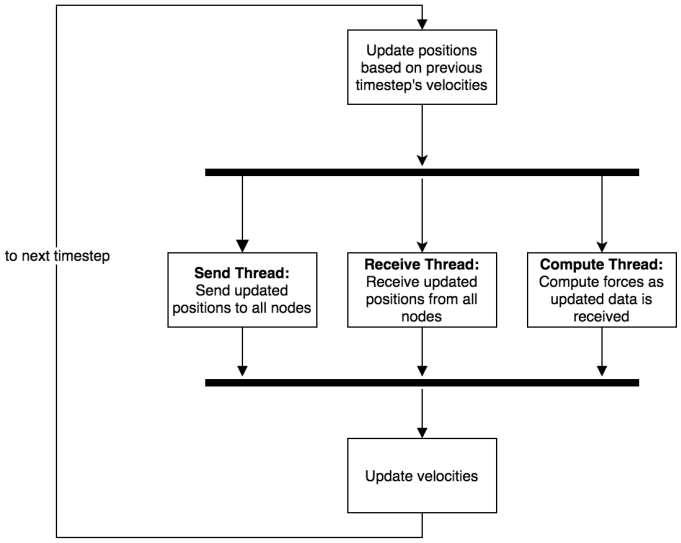
All implementations use leapfrog integration when calculating the position, velocity, and acceleration of particles at each timestep. Leapfrog integration was chosen over the traditional Euler's method because it provides second-order accuracy (versus Euler's method which is a first-order method) but still requires the same number of calculations at each step as Euler's method [5].

$$\begin{aligned} x_i &= x_{i-1} + v_{i-1/2}\,\Delta t, \\ a_i &= F(x_i), \\ v_{i+1/2} &= v_{i-1/2} + a_i\,\Delta t, \end{aligned}$$

The Leapfrog integration method where x_i represents the position at timestep i, a_i represents the acceleration at timestep i, and $v_{i+1/2}$ represents the velocity at timestep $i + \frac{1}{2}$ [5].

In the first distributed algorithm, Distributed v1, each node is given an equal sized chunk of particles. At each timestep, the positions of the particles calculated within each node's respective chunk of data, based on the previous timestep's calculated velocities, are exchanged with one another using the MPI_Allgather method..

In the second distributed algorithm, Distributed v2, in addition to dividing up the data into equal sized chunks for each node, the *pthread* library in C is utilized to separate the sending and receiving of updated positions between nodes using MPI into individual threads. The goal of this exercise was to reduce wait time while nodes exchange data; the main thread is able to continue performing force calculations with data as it becomes available.



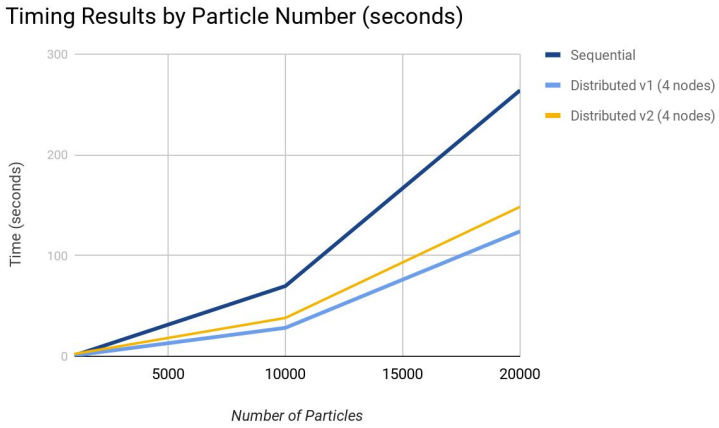
The Distributed v2 algorithm uses 3 threads during the force-calculation process.

III. Results

The use of multithreading limited our ability to test the algorithms. Unfortunately, the version of OpenMPI available in the MPAC lab does not have MPI_THREAD_MULTIPLE support. Thus, we were limited to testing the algorithms on a single machine (a MacBook Pro with a hyper-threaded quad-core 2.6GHz Intel Core i7). While this certainly limited our results in the absolute sense, it allowed us to fairly compare the performance of the multithreaded algorithm with the other algorithms.

In our tests, each algorithm was given the same set of input data (a list of particles including their masses, initial positions, and initial velocities). For the distributed algorithms, each node read the input data from a file (a possible improvement might be to have one node read the file and scatter the initial data instead). Only 4 nodes were used for the distributed tests (the maximum that can be used on a quad-core system using cores as nodes).

The times reported are real elapsed time to complete 20 timesteps of the simulation with a timestep interval of 1 second (i.e. each timestep represents 1 second).



The timing results of each algorithm (20 simulation timesteps, timestep interval = 1 sec).

IV. Conclusion

We found that while the threading present in the Distributed v2 algorithm further divided up the work, the overhead of threading led to worse performance when compared to Distributed v1. The results do however demonstrate that by distributing direct n-body simulations significant performance benefits can be achieved over sequential implementations.

References

[1] Groen, D., Portegies Zwart, S., McMillan, S. and Makino, J. (2008). Distributed N-body simulation on the grid using dedicated hardware. *New Astronomy*, 13(5), pp.348-358.

[2] Liu, P. and Bhatt, S. (2000). Experiences with parallel N-body simulation. *IEEE Transactions on Parallel and Distributed Systems*, 11(12), pp.1306-1323.

[3] D. Groen, S. P. Zwart, T. Ishiyama, and J. Makino, "High-performance gravitational N-body simulations on a planet-wide-distributed supercomputer," Computational Science & Discovery, vol. 4, no. 1, Jan. 2011.

[4] Z. Khatami, H. Kaiser, P. Grubel, A. Serio, and J. Ramanujam, "A Massively Parallel Distributed N-body Application Implemented with HPX," 2016 7th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (ScalA), pp. 57–64, Nov. 2016.

[5] "Leapfrog Integration." Wikipedia, Wikimedia Foundation, 11 Nov. 2018, en.wikipedia.org/wiki/Leapfrog_integration.