Exploring Parallelization Techniques for Gravitational N-Body Simulation Problem

Abhiroop Ghosh
Department of Electrical and Computer Engineering
Michigan State University

Abstract

N-body problem is a classical physics problem for calculating the interaction of celestial objects under a gravitational or electrical force. For a large number of bodies, an analytical solutions becomes intractable and hence, numerical methods need to be applied. However, given the large number of objects usually present within any region in the universe, a naive algorithm with quadratic complexity is not very practical. Using algorithms like Barnes-Hut in combination with parallelization methods which exploit High-Performance Computing architectures, can be a possible solution. This project aims to study the efficacy of such techniques and present a comparative analysis of the performance.

I. Introduction

N-body simulation problems [1], [2] track the behavior of multiple particles or bodies under the influence of a force, most commonly gravitational or electrical. Under the influence of such forces, the particles attain certain velocities and change their positions. For every timestep, the forces need to be recalculated and their positions updated. With a small enough time interval, this algorithm can produce results of reasonable accuracy.

Due to multiple particles interacting at the same time, this problem is extremely suited for parallelization. Distributing the particles among multiple processors [5], [6] on a high-performance computing system [1], [3] can be an efficient way to achieve fast results. For this purpose, a parallelization model combining distributed memory and shared memory paradigms has been used in this project.

The report is organized as follows. Section II describes the N-body problem and associated mathematical relations. Section III describes the numerical methods used for solving the problem. Section IV describes the parallelization scheme applied to this problem. Section V presents the results and discussion and Section VI concludes the study.

II. N-body Problem

The gravitational force (F_{ij}) between two bodies with masses m_i and m_j is given as follows,

$$\bar{F_{ij}} = \frac{Gm_i m_j}{||\bar{r_{ij}}||_2^2 + \epsilon^2} \hat{r_{ij}} \tag{1}$$

where $\bar{r_{ij}} = \bar{r_j} - \bar{r_i}$ is the vector joining the centre of masses of particles i and j. $\hat{r_{ij}}$ is the corresponding unit vector. G is the Gravitational Constant, approximately $6.674 \times 10^{-11} m^3 kg^{-1} s^{-2}$. ϵ is the softening length [2], intended to avoid singularities in the equations when two particles get very close together.

This simulation can be performed on 3D coordinates. However, for simplicity, only a 2D version is considered.

III. Numerical Methods

A naive method to perform the simulation is to loop over every pair of particles [1] and calculate the force using Eq. 1. Using the forces, we can update the velocities and particle positions using Eqs. 2 and 3. But calculation of all pairwise interactions result in an $O(N^2)$ complexity. Hence, the Barnes-Hut algorithm [1], [4] is used in this project which gives an $O(N \log N)$ complexity.

$$\bar{v_i}(t+1) = \bar{v_i}(t) + \frac{\sum_{i \neq j} \bar{F_{ij}}}{m_i} \cdot \Delta t \tag{2}$$

$$\bar{x_i}(t+1) = \bar{x_i}(t) + \bar{v_i}(t)\Delta t \tag{3}$$

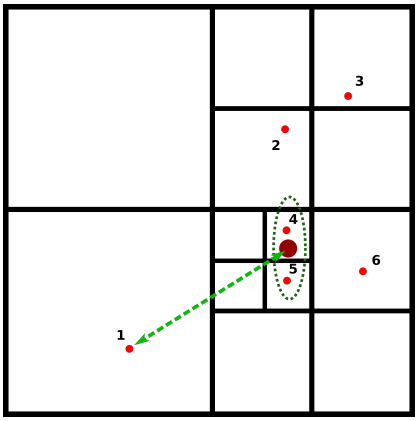

Fig. 1: Barnes-Hut Algorithm Quadrant Divisions

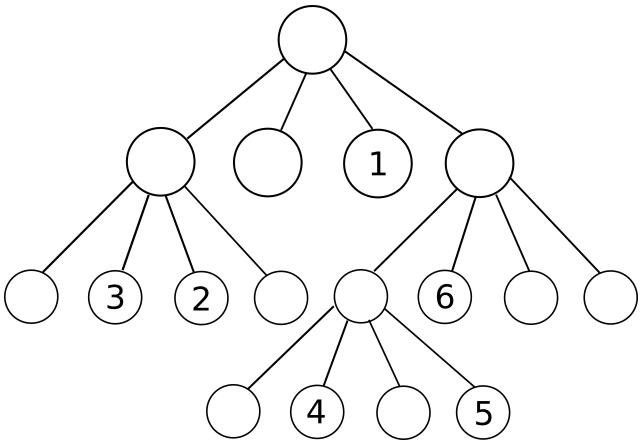

Fig. 2: Quadtree Representation

Barnes-Hut algorithm starts by subdividing the domain into quadrants recursively until every subdivision has 1 or 0 particles. One such division is shown in Fig. 1 for 6 particles. The green arrow represents how the force on particle 1 is calculated with respect to particles 4 and 5. Since 4 and 5 are sufficiently far from particle 1 and are very close together, they will be clustered together. So the centre of mass, shown in red, and the sum of their masses, are used to calculate the forces. So, instead of 15 computations, we require only 10 computations. This savings in computations increases for greater number of particles. The particles are then stored in a data structure called quadtree, shown in Fig. 2, where every node can have 4 child nodes. Each node can have at most 1 particle if they are a leaf node, 0 otherwise. Every node stores the centre of mass and the total mass of all the particles present in its children. The opening angle (θ) between a particle and a node is defined as the ratio of the node length to the distance between the particle and the centre of mass. If the opening angle is less than a pre-specified value (θ_{max}), the node can be considered as 'far' from the particle under consideration. Then, the particles contained in the children of the node can be clustered together and the centre of mass is used to calculate the force (far-field interaction). This reduces the number of computations required with only minor loss in accuracy.

The general procedure is shown as Algorithm 1.

IV. Parallelization Scheme

A combination of MPI [5] and OpenMP [6] parallelization scheme through C++ has been implemented. The implementation decomposes the particle array into multiple

Algorithm 1 Barnes-Hut N-body Simulation

Input: Number of particles (N), particle array ($\mathcal{P}$), time steps(T)
Output: Final particle coordinates ($\mathbf{R}$)

```
Initialize quadtree Q
for particle p in P do
    Insert p in Q.
for t = 1 to T do
    for particle p in P do
        for each cell c in the top level of Q do
            if c is far from p then
                Use total mass and centre of mass of c to calculate force.
            else
                Consider the children of c.
        Update velocity of p based on the force computed.
    Update positions of all particles in P based on the final calculated velocities.
```

blocks, one for each MPI rank, while taking care to maintain load balancing. Within each rank, the mathematical operations are parallelized using OpenMP threads. The only computation-intensive section of the algorithm that cannot be parallelized is the creation of the quadtree.

A. Domain Decomposition and Load Balancing using MPI

In this problem, after the creation of the tree, the particles are distributed evenly among the number of MPI ranks available by the rank 0 processor. If there are r ranks and N particles, $\lfloor \frac{N}{r} \rfloor$ particles are assigned to every processor. There might be $N - \lfloor \frac{N}{r} \rfloor$ particles remaining. Thus, to achieve optimal load balancing the first $N - \lfloor \frac{N}{r} \rfloor$ ranks are assigned one more particle. This way, no single processor has a disproportionately larger load and all particles are assigned.

Non-blocking sends and receives are applied back-to-back in the rank 0 processor, which is responsible for distributing the tasks in a correct manner. This ensures communication and computation are overlapped. After computation is done, and MPI_Waitall command is applied to wait for all outstanding computation on the other ranks to be finished.

B. OpenMP Threading

Many of the calculations within individual MPI ranks are separable. The force and velocity calculation for every particle, as well as the subsequent position update step, can be done in parallel using multiple threads. The functions compute_force() and calculate_velocity() are put in a loop iterating over all the particles in the local array. This loop is then wrapped in a *#pragma omp parallel for* directive. After this is complete, the position update process is performed in the same fashion.

The environment variable OMP_NUM_THREADS is used to control the number of threads available to each MPI Rank.

C. Parallel I/O using HDF5

The role of file I/O in this project is to record intermediate data in the simulation. This includes: the coordinates, velocity, and forces on each particle at every timestep,

time required for computation, etc. A single HDF5 file [7] is used to record the entire run. For every timestep, a new group is created, and datasets corresponding to each piece of recordable data are created. For convenience, the HDFql library [8], has been used which abstracts away some of the low-level complexities regarding HDF5 file handling.

The risk of using such a scheme is the possibility of data corruption since everything is recorded to a central file. In the future, safeguards against this, such as periodic backups, need to be incorporated.

D. Verification Test

A test case with 1000 particles was generated. A naive version of the N-body simulation problem with quadratic complexity was developed. Three unit tests are performed: (i) the quadtree module is tested to see if the particles were inserted correctly, (ii) the results of a serial Barnes Hut algorithm code are compared to the quadratic version to check if the accuracy loss is below tolerable levels, and (iii) the parallel version of the Barnes Hut algorithm code with multiple MPI ranks and OpenMP threads results should be the same as that of the serial version. These 3 tests ensure the program is working correctly.

E. Memory Usage

The primary memory consumption is due to the storage of the particle array and the quadtree in memory. Storing all the details (coordinates, velocity, etc) of a single particle in memory consumes 64 bytes. Since data is written to a file during every timestep, the same memory structures are used to store details for the next timestep. Thus, to store 50,000 particles in memory only 3 MB will be used theoretically. So the implementation is not memory-intensive.

V. Experiments and Discussion

A dataset of particles with masses ranging from 10^{10} to 10^{30} kg was used. A 2D square space of length 4×10^{10} m was used. The timestep was set at 0.01 seconds. Number of iterations was set at 1000. This means that the simulation is being run from $t = 0$ to $t = 10$ seconds. θ_{max} was set as 0.5. All bodies are assumed to be at zero velocity initially. 10 runs were performed and the average values of the time taken are reported. All experiments were run on Michigan State University's HPCC systems on the intel18 nodes.

A. Strong Scaling

For strong scaling the number of particles (N) was set to be 5000. Number of OpenMP threads was set as 1 through the OMP_NUM_THREADS environment variable. The MPI ranks were varied from 1, 2, 4, 8, 16, 32.

The results are shown in Fig. 3. It is seen that the average time per iteration or timestep is not too far from the ideal case. The increase in time in the later part can be attributed to the overhead incurred due to increased MPI communication.

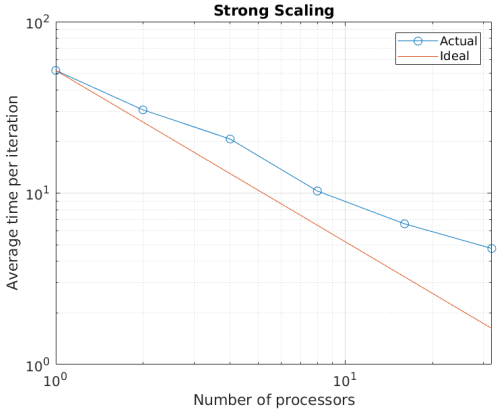

Fig. 3: Strong scaling study with 5000 particles

B. Weak Scaling

Weak scaling study is the same as the strong scaling study except the number of particles assigned to each MPI rank is kept at 1000. So for number of ranks=1, 2, 4, 8, 16, 32, the overall problem size varies as 1000, 2000, 4000, 8000, 16000, 32000.

The results are shown in Fig. 4. It can be seen that the results do not deviate too much from the ideal. For a high number of processors, the increase in MPI communication overhead account for the increased time per iteration.

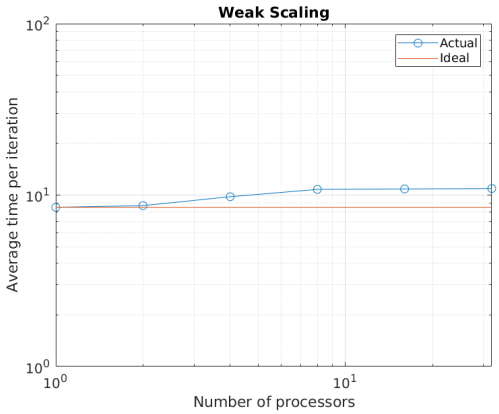

Fig. 4: Weak scaling study with 1000 particles per MPI rank

C. Thread-to-thread Scaling

In this case, the number of particles was set as 5000, but only 1 MPI rank is assigned. Number of OpenMP threads was set as 1, 2, 4, 8, 16, 32 through the OMP_NUM_THREADS environment variable.

The results are shown in Fig. 5. Time required decreases with increase in number of threads, which is expected. The deviation from the ideal curve can be due to the threading mechanism not being perfect. At some points, due to the tree traversal mechanism, there might be some variability in the computation time for the forces. This makes the amount of time required to deviate from the expected value.

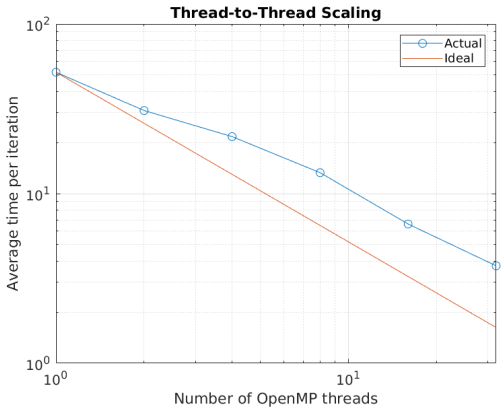

Fig. 5: Thread-to-thread speedup study with 5000 particles

VI. Conclusions

The project aimed to implement a parallelized N-body Simulation C++ code that can study the efficacy of the Barnes-Hut algorithm combined with different parallel programming paradigms. Two different parallel computation paradigms, shared memory (OpenMP) and distributed memory (MPI) were presented. They have been shown in this project to be effective in scaling the performance. When the number of threads is too large, all the scaling studies show deviations from the ideal curve mostly due to bottlenecks caused by increased MPI communication or imperfect OpenMP threading.

Future studies can focus on scaling up the problem from 2D to 3D using octrees instead of quadtrees. Studies on how much of the performance is affected by file I/O is also needed. Memory scaling studies should be done in a practical fashion through real-time testing, instead of only a theoretical analysis.

Acknowledgements

I would like to thank Prof. Sean Couch and Jared Carlson for their valuable guidance throughout the course. I would also thank my assignment group members who I was fortunate enough to work with.

References

[1] Victor Eijkhout, E. Chow, and R. van de Geijn, "Introduction to High Performance Scientific Computing," 2020.
[2] M. Trenti and P. Hut, "N-body simulations (gravitational)," *Scholarpedia*, vol. 3, no. 5, p. 3930, 2008.
[3] Victor Eijkhout, "Parallel programming for science and engineering Using MPI, OpenMP, and the PETSc library," 2020.
[4] J. Barnes and P. Hut, "A hierarchical O(N log N) force-calculation algorithm," *Nature*, vol. 324, no. 6096, pp. 446–449, 1986. [Online]. Available: https://www-nature-com.proxy1.cl.msu.edu/articles/324446a0
[5] OpenMPI, "Open MPI: Open Source High Performance Computing," 2014. [Online]. Available: https://www.open-mpi.org/
[6] OpenMP, "OpenMP," 2016. [Online]. Available: https://www.openmp.org/
[7] The HDF Group, "The HDF5® Library & File Format - The HDF Group," 2020. [Online]. Available: https://www.hdfgroup.org/solutions/hdf5/
[8] "HDFql: The Easy Way to Manage HDF5 Data." [Online]. Available: https://www.hdfql.com/