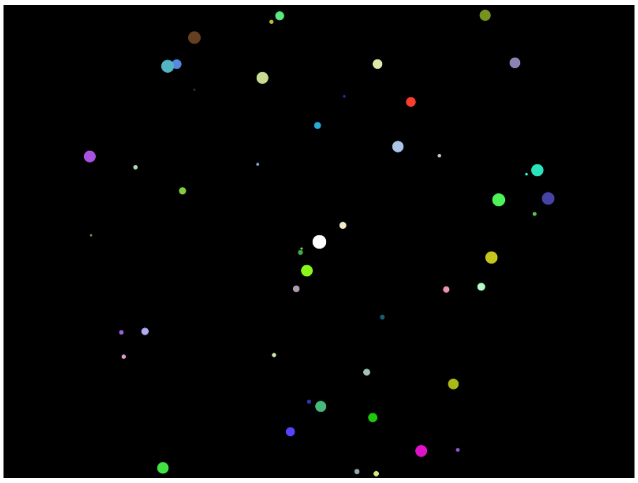
École Polytechnique

CSE305 - N Body Simulations
Final Report



Author:

Julien Girod, Liam Loughman

Academic year 2023/2024

1 Introduction

The N-Body problem is a classical problem in physics and astronomy, involving the prediction of individual motions of a group of celestial objects interacting with each other. These simulations are critical for understanding various astrophysical phenomena and are widely used in fields such as cosmology, galaxy formation, and stellar dynamics.

N-Body simulations have a rich history, starting from early analytical attempts and evolving with the advent of computational methods. Initially, simulations were limited by computational power, allowing only a small number of particles to be simulated. With the progression of technology, particularly the development of fast algorithms and high-performance computing, modern simulations can handle billions of particles, providing deep insights into the large-scale structure of the universe.

Several key techniques have been developed to improve the efficiency of N-Body simulations. Direct methods compute forces without approximation, offering high accuracy but at a significant computational cost, scaling with $O(n^2)$. Tree codes, such as the barnes hutt algorithm, reduce this complexity to $O(n \log n)$ by approximating the forces from distant particles, making them suitable for simulations involving a large number of particles.

This report details the implementation and performance evaluation of various N-Body simulation algorithms, including both sequential and parallel approaches, as well as the barnes hutt algorithm. The aim is to compare the efficiency and accuracy of these methods under different conditions and to highlight the advantages of parallel computing in handling large-scale simulations.

2 Algorithms

In this section, we discuss the various algorithms implemented to solve the N-Body problem. The algorithms range from straightforward sequential methods to more complex parallel implementations and the efficient barnes hutt algorithm.

2.1 Sequential Naive

The sequential naive algorithm is the most straightforward approach to solving the N-Body problem, wherein the gravitational forces between every pair of bodies are computed directly. This method involves iterating through each pair of bodies and computing the gravitational force exerted on each other. The computational complexity of this approach is $O(N^2)$, making it simple but not efficient for large systems.

The force between two bodies is determined using Newton's law of universal gravitation. For each pair of bodies i and j, the force $F_{i,j}$ is calculated as:

$$F_{i,j} = G\frac{m_i m_j}{r_{ij}^2}$$

where G is the gravitational constant, m_i and m_j are the masses of the bodies, and r_{ij} is the distance between them. This force is then decomposed into its x and y components, which are used to update the velocities and positions of the bodies.

2.2 Parallelization

Parallelization techniques are crucial for enhancing the performance of N-Body simulations, especially when dealing with large numbers of bodies. By distributing the computational load across multiple processors, these techniques can significantly reduce the time required for simulations. In this section, we describe the various parallelized functions implemented in the project.

- The parallel step simulation algorithm consists of first computing the forces on each body in a non-parallel, sequential manner, and then distributing the task of updating the velocities and positions of the bodies across multiple threads. Each thread updates a distinct subset of bodies, ensuring that no two threads update the same body simultaneously.
- In the parallel forces simulation, the computation of the force itself is parallelized. The bodies are divided into chunks, with each thread responsible for computing the forces on its specific chunk of bodies, ensuring that the force on each body is computed only once. After computing the forces, we update the velocities and positions of the bodies. This step is non-parallel, sequential.
- The parallel combined simulation algorithm integrates both parallel force calculation and parallel position update. This method aims to maximize the use of parallel computing resources by splitting both the force calculation and the update phases across multiple threads.The bodies are once again divided into chunks, with each thread responsible for both computing the forces and updating the velocities and positions of the bodies in its chunk.

2.3 Barnes Hutt

The barnes hutt algorithm is an efficient method for solving the N-Body problem by approximating the interactions between bodies, thus reducing the computational complexity from $O(N^2)$ to $O(N \log N)$. This algorithm is particularly useful for large-scale simulations where computing the forces between all pairs of bodies would be computationally expensive.

The first step is to construct the quadtree. The simulation space is recursively divided into quadrants, and each body is inserted into the appropriate quadrant based on its position. This process continues until each body is in a separate leaf node. Once the tree is constructed, the algorithm computes the total mass and center of mass for each node. For each body, the algorithm traverses the tree to compute the net gravitational force acting on it. If a node is sufficiently far from the body (determined by a threshold parameter θ), the node is treated as a single body located at its center of mass. Otherwise, the node is further subdivided, and the process continues with its children. This reduces the number of force computations by approximating the effects of distant bodies. After computing the forces, the positions and velocities of the bodies are updated based on the computed forces.

Algorithm 1 Barnes-Hut Algorithm Simplified Pseudocode

```
procedure BarnesHuttSimulation(bodies, root, dt)
    root.computeMassDistribution()
    for body in bodies do
        Initialize force to zero (fx, fy)
        root.computeForce(body, fx, fy)
        Update body's velocity based on force
        Update body's position based on velocity
    end for
end procedure

procedure Insert(b)
    if node is a leaf then
        if no body stored in node then
            Store body b in the node
        else
            Subdivide node into four quadrants (NW, NE, SW, SE)
            Redistribute current body and insert new body b
        end if
    else
        Determine appropriate quadrant for b
        Insert b into that quadrant
    end if
end procedure

procedure ComputeMassDistribution
    if node is not a leaf then
        Recursively update mass and center for each quadrant
        Calculate combined mass and center of mass for the node
    end if
end procedure

procedure ComputeForce(body, fx, fy)
    Calculate distance to body
    if distance is sufficiently large or node is a leaf then
        Approximate force from node and update fx, fy
    else
        Recursively compute force contribution from each quadrant
    end if
end procedure
```

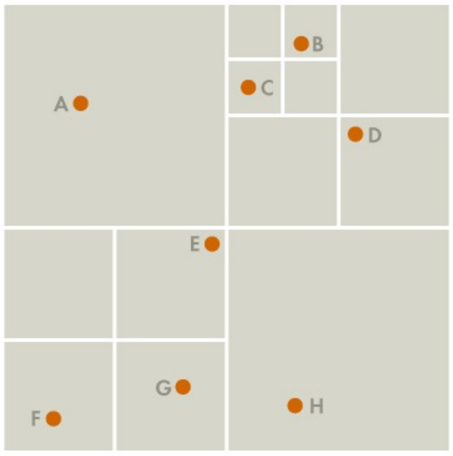


Figure 1: The space is recursively subdivided into quadrants until each subdivision contains 0 or 1 bodies.

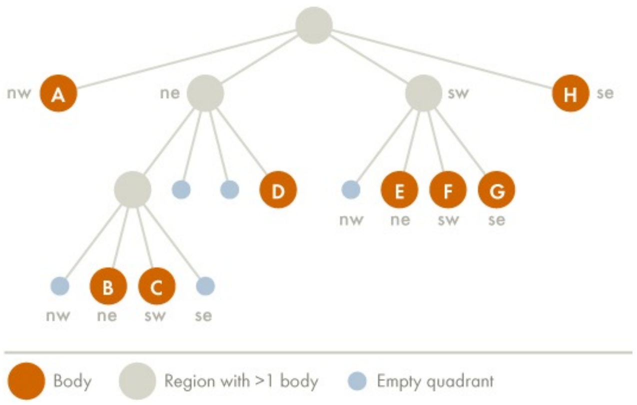


Figure 2: Resulting tree structure.

2.4 Collision

The collision algorithm detects and resolves collisions by adjusting both the velocities and positions of the interacting bodies. It begins by computing the distance between each pair of bodies and checks if this distance is less than the sum of their radii, indicating a collision. If a collision is detected, the algorithm computes the normal vector between the bodies to determine the direction of the collision. It then computes the relative velocity along this normal vector to assess whether the bodies are moving towards each other. If they are, the algorithm computes an impulse based on their masses and relative velocities to update their velocities, ensuring momentum conservation. Finally, to prevent overlapping, it corrects their positions by moving them apart along the normal vector by half the overlap distance.

3 Performance of Algorithms

To evaluate the performance of each N-Body algorithms, we conducted several performance tests on the lab machine which is equipped with an Intel(R) Xeon(R) $E-2246G$ CPU @ $3.60G$Hz, featuring 6 cores and 12 threads. It has 62 GiB of RAM and a Quadro $P2200$ GPU with 5120 MiB of memory.

We ran a series of tests with varying numbers of bodies: 1, 10, 100, 1000, and 10000. For each number of bodies, we measured the execution time using different numbers of threads to evaluate how well the algorithms scale with increased parallelism.

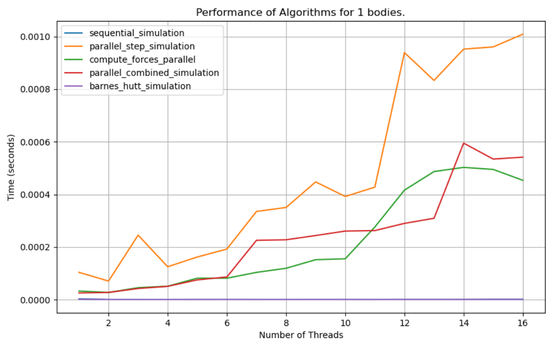


Figure 3: Performance results for 1 body.

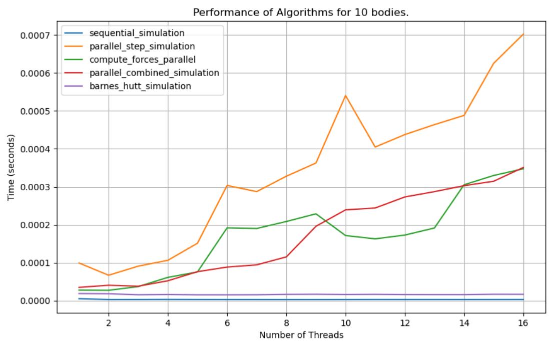


Figure 4: Performance results for 10 bodies.

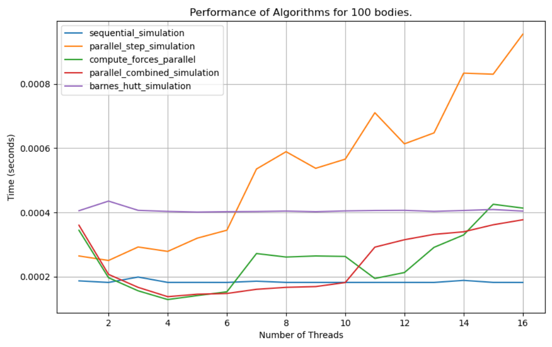


Figure 5: Performance results for 100 bodies.

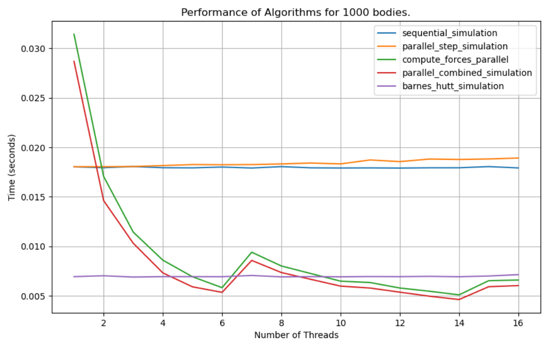


Figure 6: Performance results for 1000 bodies.

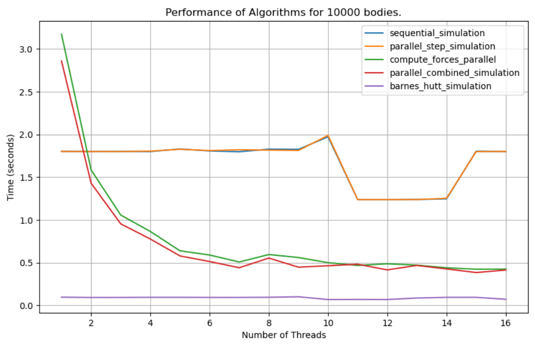


Figure 7: Performance results for 10000 bodies.

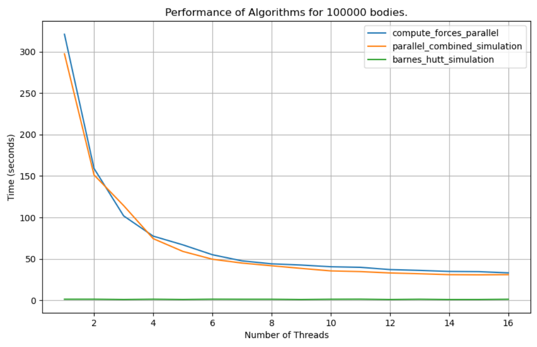


Figure 8: Performance results for 100000 bodies.

In all 5 plots, the execution time of the sequential simulation and barnes hutt simulation algorithms remain constant irrespective of the number of threads, which is expected since these algorithms do not leverage parallel processing.

In Figure 3 and 4: The parallel step simulation algorithm shows an increase in execution time as the number of threads increases. This suggests that for a single body, the overhead associated with managing multiple threads outweighs any potential performance gains from parallelization. The compute forces parallel and parallel combined simulation algorithms also exhibit increased execution times with more threads, but to a lesser extent compared to the parallel step simulation algorithm. This indicates that while there is overhead, it is not as pronounced. However, the results still highlight that for very small problem sizes, parallelization does not provide a performance benefit and can even be detrimental due to overhead.

In Figure 3, the execution times for the sequential simulation and the barnes hutt simulation algorithms are nearly identical. This indicates that for a single body, both algorithms perform equally well, with negligible differences in computational time. In Figure 4, however, the sequential simulation algorithm has slightly faster execution times compared to the barnes hutt simulation. This minor difference suggests that while both algorithms are efficient for small problem sizes, the sequential approach may have a slight edge in scenarios involving minimal computational complexity.

In Figure 5: The sequential simulation algorithm exhibits significantly faster execution times compared to the barnes hutt simulation. In the Barnes-Hut algorithm, the observed slower performance for 100 bodies is primarily due to the overhead associated with constructing and maintaining the tree data structure, where the benefits of faster computation from this structure do not yet offset the initial setup costs.

The parallel step simulation still exhibits an execution time that increases as more threads are used, suggesting that the overhead outweighs the benefits of parallelization, leading to reduced efficiency.

The compute forces parallel and parallel combined simulation algorithms demonstrate better scalability compared to the parallel step simulation. Execution time decreases as the number of threads increases up to 8 threads, after which the performance gains slightly degrade with higher niumber of threads. This indicates that the algorithm benefits from parallelization but is also subject to overhead for higher number of threads.

In Figure 6 and 7: For the sequential simulation algorithm, the execution time remains relatively constant across all thread counts, which is expected since this algorithm does not leverage parallel processing.

The parallel step simulation maintains a consistent execution time across different thread counts for both 1000 and 10000 bodies. This indicates that parallelizing the $O(n)$ component of the algorithm is less impactful in the context of the more dominant $O(n^2)$ force computation step, resulting in a runtime that closely mirrors the sequential execution.

The compute forces parallel and parallel combined simulation both demonstrate significant reductions in execution time as the number of threads increases up to around 8 threads for both 1000 and 10000 bodies. Beyond this point, the performance gains plateau due to the overhead of additional threads.

The barnes hutt simulation maintains a consistently low execution time across all number of threads in both plots. In Figure 6, for 1000 bodies, the barnes hutt simulation (purple line) is outperformed by both the compute forces parallel and parallel combined simulation algorithms. However, in Figure 7, for 10000 bodies, the barnes hutt simulation demonstrates the best performance. This highlights the barnes hutt algorithm's efficiency and scalability for larger problem sizes.

In figure 8, the barnes hutt simulation maintains a remarkably low execution time across all number of threads, demonstrating its superior efficiency and scalability for large-scale simulations. Both the compute forces parallel and parallel combined simulation algorithms show improved execution times with increasing threads up to 8, after which the benefit of adding more threads diminishes, suggesting an optimal thread count for these algorithms.

4 Accuracy of Algorithms

To evaluate the accuracy of the various N-Body simulation algorithms, we compare their results against the sequential simulation algorithm which we use as our baseline due to its straightforward approach. To check the accuracy, we plot the trajectories of three randomly selected bodies and compute the mean positional error of all bodies for each algorithm compared to the sequential simulation.

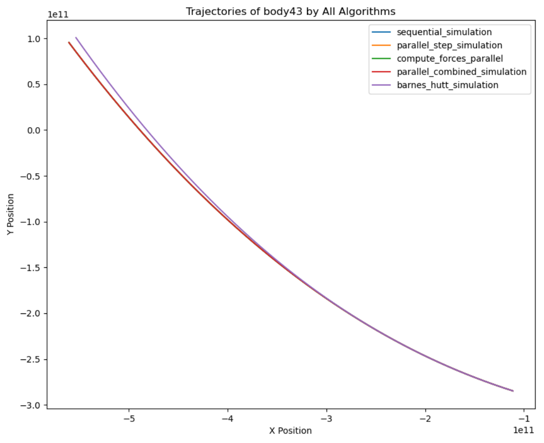


Figure 9: Trajectories of body9 computed by all algorithms.

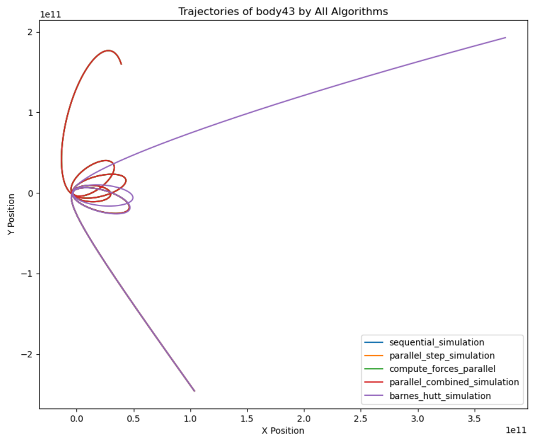


Figure 10: Trajectories of body22 computed by all algorithms.

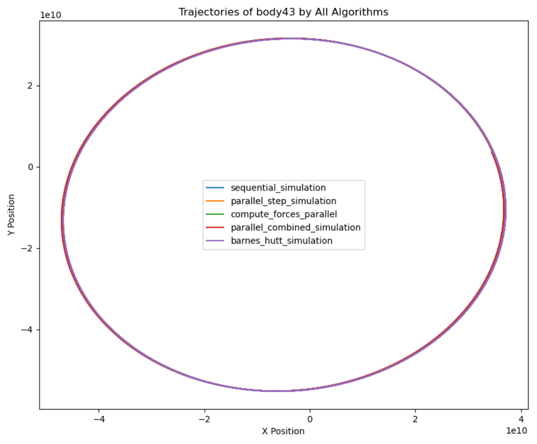


Figure 11: Trajectories of body40 computed by all algorithms

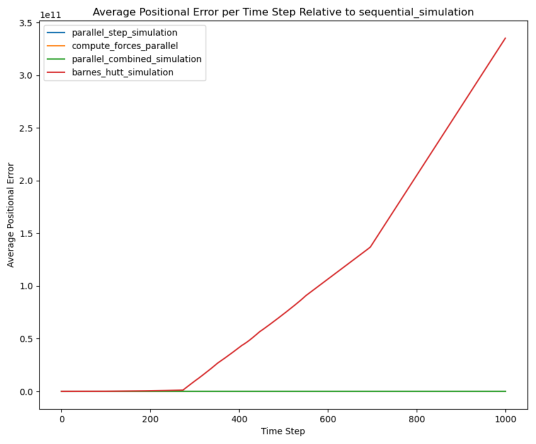


Figure 12: Mean positional error of all algorithms compared to sequential simulation algorithm.

In figures 9, 10, and 11, we observe that the barnes hutt simulation algorithm deviates noticeably from the parallel combined simulation, while the sequential simulation, parallel step simulation, and compute forces parallel algorithms overlap with one of the parallel combined simulation or barnes hutt simulation algorithms.

The primary reason for the deviation observed in the barnes hutt simulation is its use of spatial decomposition to approximate the gravitational forces. Instead of calculating the force between every pair of bodies, barnes hutt groups distant bodies into clusters and approximates their combined gravitational effect using the center of mass of each cluster.

The same is observed in figures 12: the barnes hutt simulation algorithm deviates noticeably from the parallel combined simulation, while the sequential simulation, parallel step simulation, and compute forces parallel algorithms overlap with one of the parallel combined simulation or barnes hutt simulation algorithms, which confirms our observation in the trajectories.

To better understand the accuracy of the sequential simulation, parallel step simulation, and compute forces parallel algorithms, we plot the trajectories of the same bodies and the mean positional error, this time omitting the barnes hutt algorithm.

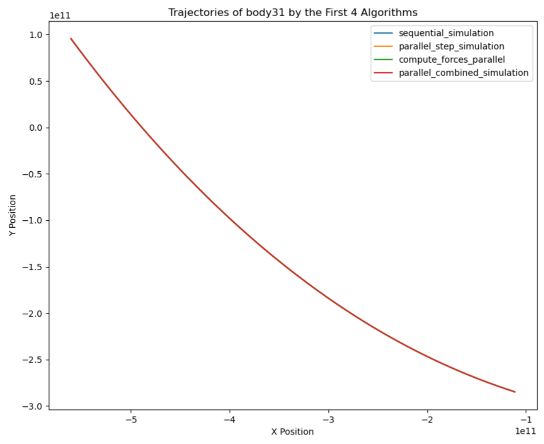


Figure 13: Trajectories of body9 computed by all step simulation, compute forces parallel, and parallel combined simulation algorithms.

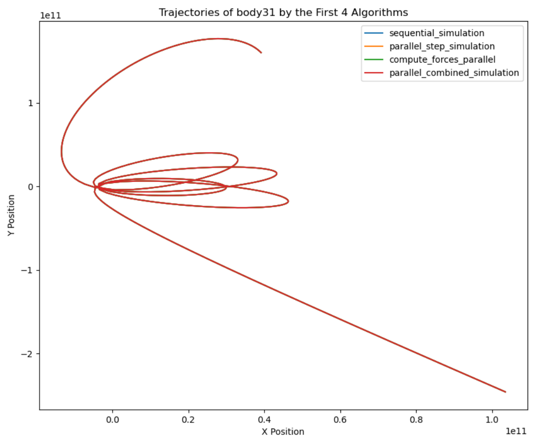


Figure 14: Trajectories of body22 computed by all step simulation, compute forces parallel, and parallel combined simulation algorithms.

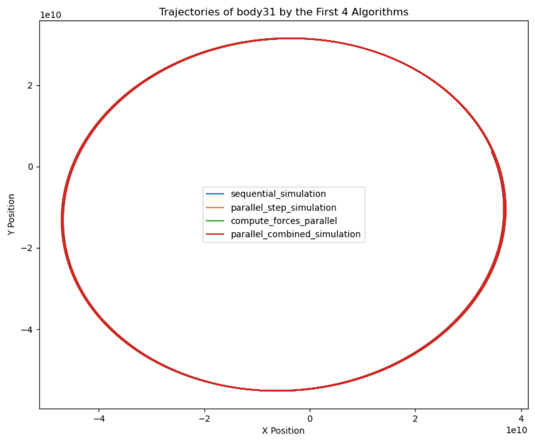


Figure 15: Trajectories of body40 computed by all step simulation, compute forces parallel, and parallel combined simulation algorithms

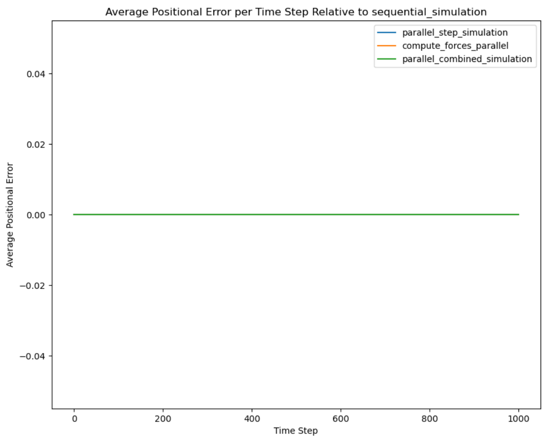


Figure 16: Mean positional error of parallel step simulation, compute forces parallel, and parallel combined simulation algorithms compared to sequential simulation algorithm.

In figures 13, 14, and 15, the trajectories of the sequential simulation, parallel step simulation, compute forces parallel, and parallel combined simulation are overlapping. In figure 16, all three algorithms exhibit minimal positional error, essentially overlapping with each other and showing a flat line at zero.

This indicates that the sequential simulation, parallel step simulation, compute forces parallel, and parallel combined simulation algorithms produce nearly identical trajectories and maintain high accuracy with minimal positional error. This consistency suggests that the parallelization techniques used do not compromise the accuracy of the simulations, making these algorithms reliable for n-body simulations.

5 Performance and Accuracy of Barnes-Hutt with respect to θ

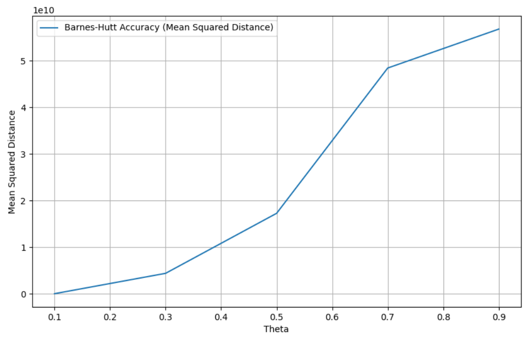


Figure 17: Error as a function of Theta

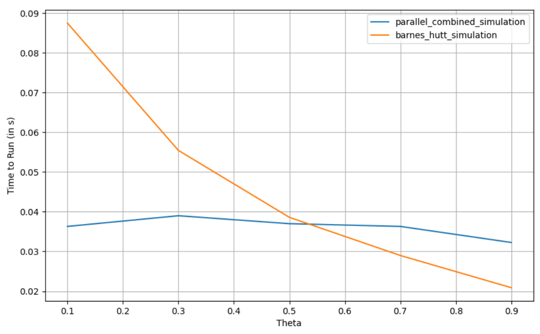


Figure 18: Runtime as a function of Theta

The graphs illustrate the trade-off between accuracy and runtime in the Barnes-Hutt algorithm as influenced by the theta parameter. As theta increases, the algorithm performs more approximations, which in turn increases its speed as shown in Figure 18. However, these approximations also result in a higher error, as evidenced by the rising mean squared distance in Figure 17. This demonstrates the fundamental trade-off between achieving faster computational times and maintaining high accuracy in simulations, where selecting an appropriate theta value is crucial depending on the application's tolerance for error versus the need for speed.

6 Parallelization of the Barnes-Hut Algorithm

To enhance the efficiency of the Barnes-Hut algorithm in our simulations, we propose several strategies aimed at better utilizing multicore processors and reducing computational overhead:

1. **Parallel Tree Construction:** Divide the simulation space into distinct regions and assign the construction of different parts of the quad-tree to multiple threads. This approach can significantly diminish the time required for tree construction.

2. **Concurrent Force Computation:** Allocate groups of bodies to different threads such that each thread independently computes the gravitational forces using the quad-tree. This method leverages parallel processing to speed up the force computation phase.

3. **Kernel Design:** Develop CUDA kernels for essential operations like tree construction and force computation. Optimize these kernels to enhance GPU performance by minimizing thread divergence and maximizing core utilization.

4. **Optimize Tree Updates:** To save computational efforts, consider updating the quad-tree few steps instead of every single step. This strategy can introduce minor inaccuracies but can substantially reduce computational load.

5. **Thread Synchronization Optimization:** Minimize the overhead associated with thread synchronization by reducing the scope of critical sections. Utilizing finer-grained locks or adopting lock-free structures can help in achieving this.

These strategies are designed to optimize both the computational and temporal efficiency of the Barnes-Hut algorithm, ensuring that the simulation can scale effectively with the number of processor cores.

7 Conclusion

This project has explored various N-Body simulation algorithms, ranging from the traditional sequential approach to more sophisticated parallel techniques and the barnes hutt algorithm. Our findings highlight the nuanced trade-offs between computational efficiency, scalability, and accuracy inherent in each method, particularly under varying conditions of system size and computational resources.

The sequential approach, while simple and highly accurate, is computationally intensive and less feasible for large-scale simulations. In contrast, the barnes hutt algorithm demonstrated superior scalability, efficiently handling large numbers of bodies by approximating distant interactions and significantly reducing computational complexity. This makes it particularly suited for extensive simulations typical in astrophysical studies.

Parallel algorithms also showed promising results by effectively utilizing multiple processors to expedite computations. The performance evaluation revealed that while parallelization introduces overhead, particularly for smaller systems, it significantly enhances performance as the number of bodies increases.

The accuracy assessments confirmed that while the barnes hutt algorithm introduces some deviations due to its approximations, it remains a powerful tool for large-scale simulations where slight inaccuracies are a worthwhile trade-off for improved performance.

In conclusion, the choice of algorithm should be dictated by the specific requirements of the simulation task at hand, balancing the need for accuracy with computational resources and time constraints.

References

[1] Josh Barnes and Piet Hut. A hierarchical o(n log n) force-calculation algorithm. *Nature*, 324:446–449, 1986.

[2] Barnes-hut algorithm. http://arborjs.org/docs/barnes-hut.

[3] Wikipedia contributors. Barnes–hut simulation. https://en.wikipedia.org/wiki/Barnes%E2%80%93Hut_simulation.

[4] Imagemagick. https://imagemagick.org/Magick++/.

A Appendix

A.1 Responsibilities of Team Members

Julien Girod implemented the initial sequential force and worked on parallelizing the force computation and later added the barnes hutt algorithm. He also implemented the random body generator and integrated runtime measurement functionalities. Julien made further improvements to the simulation, including fixing the barnes hutt method, adjusting colors and sizes for bodies, and fixing issues with the draw frame and body generation functions. Liam Loughman implemented the other parallel functions along with the collision function. He also integrated rendering capabilities using the Magick++ library and developed the infrastructure for testing performance, including adding a Makefile and a plotting script.