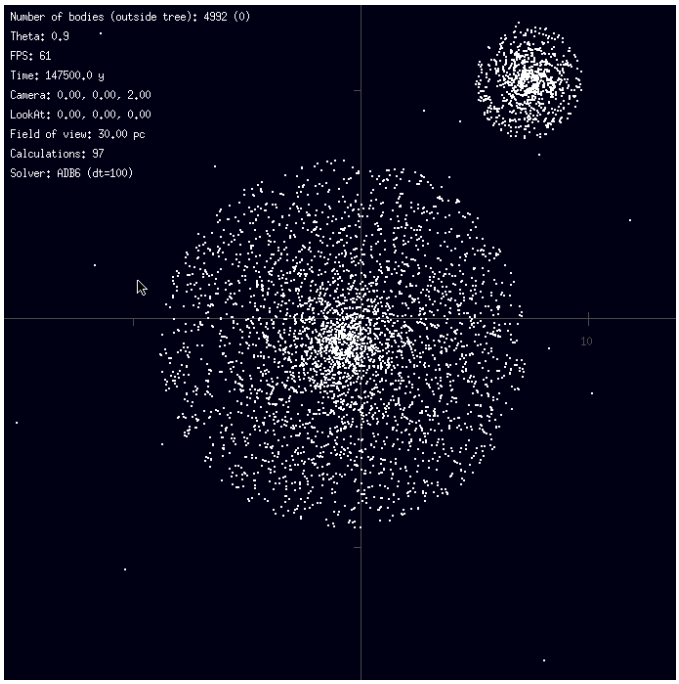
Parallelization of Tree-Particle Mesh Algorithm for N-Body Galaxy Simulations

15418 Course Project
Spring 2024
Kaiwen Geng and Alex Nguyen
Project Website



Contents

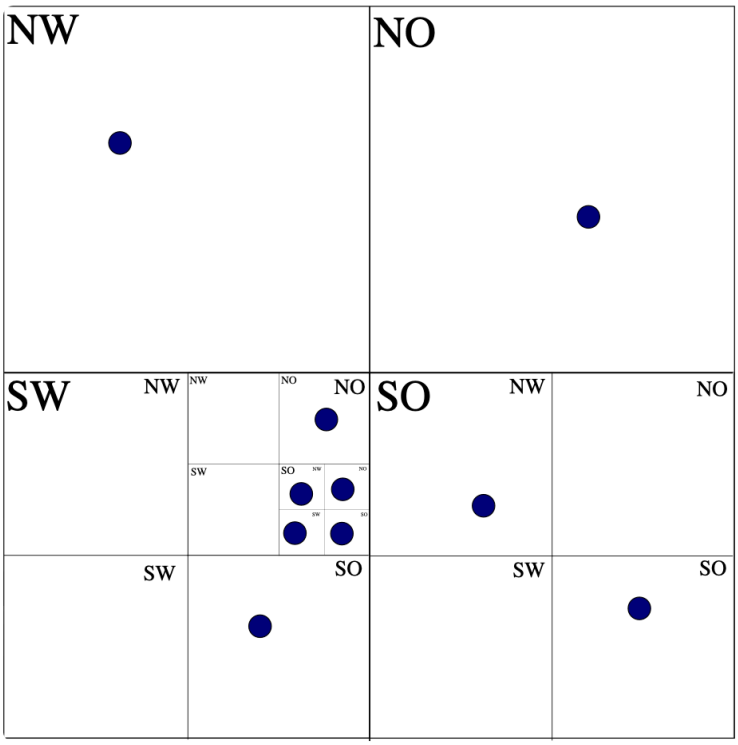
1. Summary

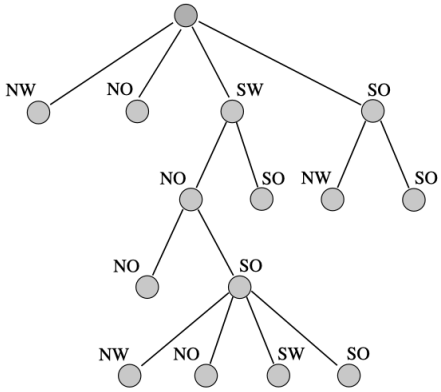
In this project, our goal was to create an N-body galaxy simulation by implementing a Tree-Particle Mesh algorithm which combined a Barnes-Hut Tree algorithm with a Particle Mesh algorithm to utilize the advantages of each algorithm in force calculations between particles in space. We intended to utilize the Barnes-Hut algorithm to perform force calculations for particles in close proximity (in high density regions) and Particle Mesh algorithm to perform force calculations on a particle.

2. Background

2.1 The Barnes-Hut (Tree) Algorithm

The first of the algorithms that we utilize is the Barnes Hut algorithm. The defining feature of a Barnes-Hut algorithm is its use of a quadtree to sort the particles of the simulation by spatial proximity, with each node containing either particles or additional leaf nodes containing particles.





Once the quadtree has been constructed and all particles have been placed within the data structure, the forces on each particle are calculated by traversing the quadtree. The force

calculations are performed using several mathematical equations based on Newton's Law of Universal Gravitation, which we will not talk about here because they are not relevant to the advantages of this algorithm.

For nodes that are sufficiently far away from the particle in question, the group of particles in a distant node is treated as a single entity with their combined mass located at their center of mass. This approximation greatly reduces the number of interactions that need to be computed. It is this approximation that we aim to solve with the Particle Mesh algorithm rather than the Barnes Hut tree to increase the accuracy of our simulation with regards to the forces of far away particles acting on a particle in question.

2.2 The Particle Mesh Algorithm

The Particle Mesh algorithm is a computational method primarily used in physics simulations to solve partial differential equations that appears in various fields like fluid dynamics, electromagnetism, and astrophysics. At its core, the Particle Mesh algorithm combines two main techniques: particle methods and mesh-based methods. Particle methods represent the system by discrete particles, while mesh-based methods discretize space into a grid. The Particle Mesh algorithm integrates these by employing particles to carry information about the system's behavior and a mesh to efficiently calculate interactions between particles.

The algorithm follows several main steps:

1. Particle Representation: The system is represented by a collection of particles, each possessing properties like position, velocity, and other relevant attributes depending on the problem domain.
2. Mesh Construction: A mesh, typically a grid, is superimposed over the domain where particle interactions occur. This mesh provides a structured framework for efficiently calculating interactions between particles.
3. Particle-Mesh Interaction: The particles' attributes, such as charge or density, are deposited onto the mesh using interpolation techniques. This step spreads the influence of each particle across nearby mesh points, allowing for smooth interactions.
4. Mesh Calculations: Once the particles' attributes are mapped onto the mesh, traditional numerical techniques like finite difference or finite element methods are applied to solve the PDEs governing the system's dynamics. These calculations account for interactions between particles as well as their surroundings.
5. Updating Particle Properties: After solving for the mesh-based equations, the updated values are transferred back to the particles using interpolation. This step ensures that the particles reflect the changes in the system accurately.
6. Iteration: The process iterates over time steps, advancing the simulation by recalculating particle interactions and mesh-based computations.

2.3 The Tree-Particle Mesh Algorithm

The Tree-Particle Mesh (TPM) algorithm is a computational technique used primarily in astrophysics and cosmology to simulate the gravitational interactions among a large number of particles in a cosmological context, such as in the formation of galaxies and large-scale structures in the universe. At its core, TPM combines two main methods: the Particle-Mesh (PM) method and the Tree method. The PM method represents the distribution of particles by interpolating their mass onto a regular grid, simplifying calculations by converting the continuous mass distribution into a discrete one. TPM combines these methods to overcome their individual limitations. It divides space into a grid for the PM calculation but also constructs a hierarchical tree for efficient computation of gravitational forces. The grid is used to compute forces for particles that are relatively far away from each other, while the tree is used to calculate forces for particles that are close.

The algorithm proceeds in several steps:

1. Construct a hierarchical tree based on the distribution of particles.
2. Interpolate particle masses onto a grid.
3. Compute gravitational forces using the PM method for distant particles.
4. Compute forces for nearby particles using the hierarchical tree.
5. Combine the forces obtained from both methods to determine the overall gravitational interactions.

TPM offers several advantages. It efficiently handles large particle numbers, allowing simulations of complex cosmological scenarios. Additionally, it balances accuracy and computational cost by employing different methods depending on the particle distribution. However, TPM also has limitations. It requires careful tuning of parameters to achieve optimal performance, and the accuracy of the results may vary depending on the specific implementation.

3. A Naive Model

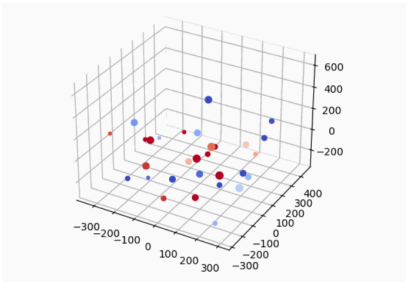
3.1 A Naive Model Based on Coulomb's law without TPM

In this naive N-body simulation, each particle inhabiting the three-dimensional space possesses several defining attributes, including mass, charge, position, and initial velocity. These particles dynamically interact with one another through the fundamental forces of nature, which is primarily governed by Coulomb's law for electrostatic interactions and Newton's law of universal gravitation for gravitational effects, though with a relatively minor influence compared to electrostatic forces. The computation of these interactions involves meticulous calculations of the

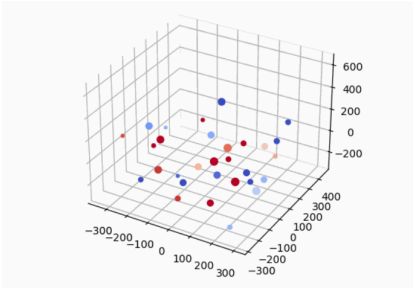
forces exerted between each pair of particles, accounting for their respective charges and distances. Coulomb's law describes the force of attraction or repulsion between charged particles, where particles with opposite charges attract each other, while particles with like charges repel. The strength of these interactions varies proportionally with the magnitude of the charges and inversely with the square of the distance between them. Furthermore, gravitational interactions, although relatively weaker compared to electrostatic forces in this simulation, contribute to the overall dynamics of the system. Newton's law of universal gravitation governs the gravitational attraction between particles, where each particle exerts an attractive force on every other particle,and is dependent on their masses and the distance separating them. Upon computing the forces acting on each particle, Newton's second law of motion comes into play. The net force experienced by a particle is related to its resulting acceleration. By integrating these accelerations over time, the simulation calculates the updated velocities and positions of the particles, thereby simulating their trajectories and interactions throughout the simulated timeframe.

To aid visualization and comprehension of the system's behavior, the particles are color-coded based on their charges, with negatively charged particles depicted in shades of blue and positively charged particles in shades of red. The intensity of these colors corresponds to the magnitude of the respective charges, allowing observers to discern the distribution of charges within the system visually. Moreover, the size of each particle's representation in the visualization is proportional to its mass, providing additional insights into the relative masses of the particles and their impact on the dynamics of the system.

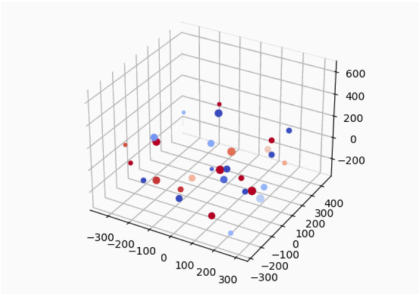
For experimentation and analysis, a specific scenario has been set up, wherein 30 particles with randomly generated attributes—positions and charges—are introduced into the three-dimensional space. The positions of these particles are recorded at discrete time intervals (0, 75, 150, and 225, corresponding to increments of 0.0025 seconds), enabling the observation and study of their evolving trajectories and interactions over time.



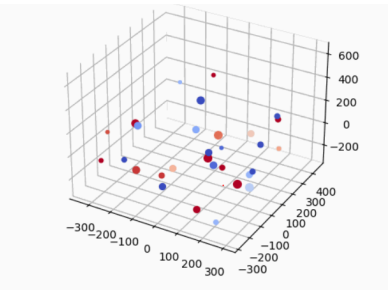
(a)



(b)



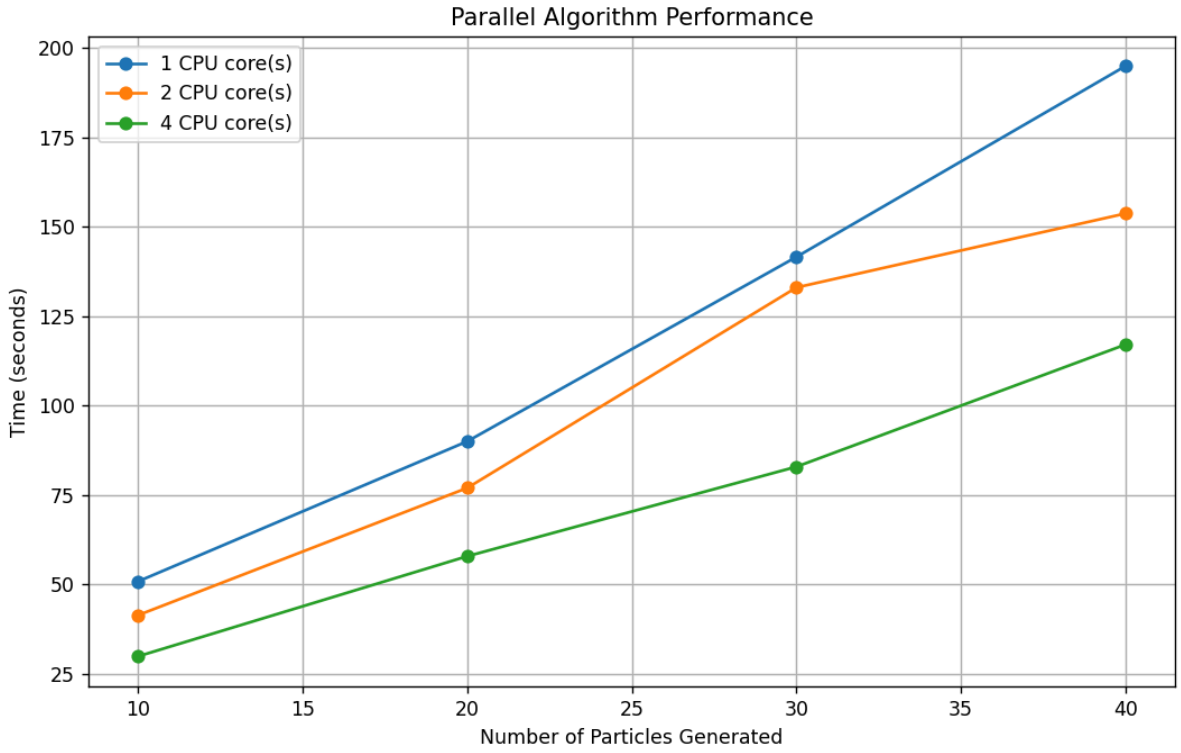
(c)



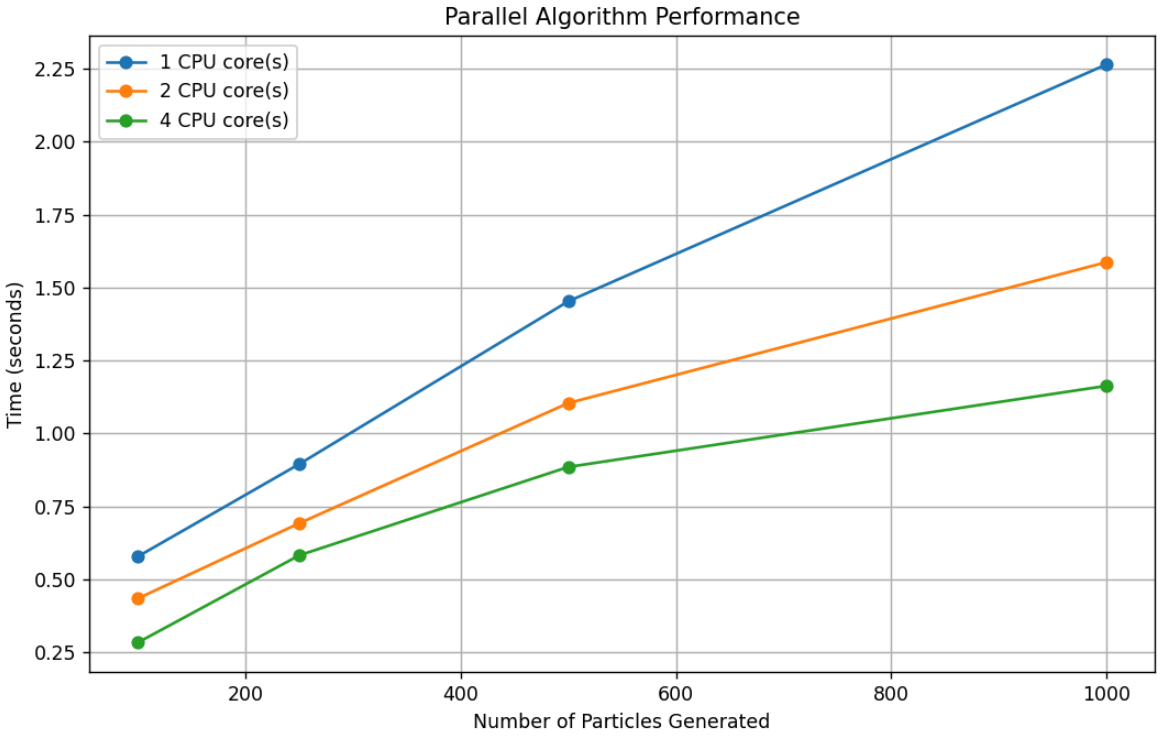
(d)

3.2. Parallelization Strategy for Naive Model

3(a)



3(b)



The plot above is the execution time of our program and is measured based on wall-clock time in seconds. We experiment with generating 10 to 40 random particles (full simulations, plot 3a) and force calculation only with 100, 250, 500, and 100 particles (plot 3b)

Parallelization in this simulation is achieved using OpenMP directives, which enable the concurrent execution of certain tasks across multiple threads. The key components of the simulation where parallelism is used include:

1. Force Calculation: The computation of forces between particle pairs is parallelized. This allows for simultaneous calculation of forces between multiple pairs of particles, so we can leverage the multi-core architecture of modern processors to improve computational efficiency.
2. Integration: The integration step, which updates the velocities and positions of particles based on the computed forces, is also parallelized. By distributing the integration calculations across multiple threads, the simulation can effectively handle a large number of particles and time steps.
3. Particle Generation: The process of generating random particles is parallelized to expedite the initialization of the simulation. Multiple particles can be generated concurrently, reducing the overall setup time for the simulation.

3.3 Analysis of Non-Linear Speedup for Naive Model:

1. Output File Writing Overhead:

One significant reason for non-linear speedup is the overhead associated with writing simulation results to output files. In the parallel simulation, each thread may attempt to write to the output file simultaneously at the end of every iteration. This concurrent access to the file system can lead to contention and overhead, especially when multiple threads contend for the same resource. As the number of threads increases, the contention for file access escalates, which can potentially cause delays and inefficiencies. This contention can result in a bottleneck, limiting the overall speedup achieved by parallelization.

2. Synchronization Overhead:

Another potential reason for non-linear speedup is the overhead incurred due to synchronization mechanisms employed by OpenMP directives. In the parallel sections of the code, synchronization primitives such as barriers and locks may be utilized to coordinate the execution of threads and ensure data consistency. The use of these synchronization mechanisms introduces additional overhead, as threads may need to wait for each other to complete certain operations or access shared resources. This overhead can become more obvious as the number of threads increases, leading to diminishing returns in terms of speedup.

3. Load Imbalance:

Load imbalance among threads can also contribute to non-linear speedup. In the N-body simulation, certain computational tasks, such as force calculation and integration, may not be evenly distributed among threads due to variations in particle densities or interaction patterns. Threads may finish their assigned workloads at different times, leading to idle time for some threads while others continue to compute. This idle time reduces the overall efficiency of parallel execution and diminishes the speedup achieved.

4. Memory Bandwidth Limitations:

Memory bandwidth limitations can also impact the speedup of the parallel simulation. As the number of threads increases, the demand for memory bandwidth escalates due to concurrent access by multiple threads. This increased demand can saturate the available memory bandwidth, leading to contention and delays in memory access. Consequently, the overall performance improvement may not scale linearly with the number of threads, as memory access becomes a limiting factor.

4. Combining Algorithms

4.1 Approach

We began with Barnes-Hut starter code based on an implementation by beltoforion. As stated before, we intended to utilize the Particle Mesh algorithm for calculating the gravitational potential over large scales where interactions can be effectively averaged over large distances, and the Barnes-Hut algorithm for more accurate, direct calculations of interactions between particles that are in relatively close proximity, where the detailed structure and dynamics are more important.

4.1.1 Parallelization Strategy for TPM

We had several goals for parallelization of the Tree Particle Mesh algorithm, namely:

Particle Decomposition: This approach would simplify data management but could lead to uneven computational loads if particle distribution becomes non-uniform. Additionally we aimed to utilize dynamic load redistribution during the reconstruction of the tree and particle mesh after significant particle movement changes affect the particle distribution.

Sharing Particle Data: We aimed to use openmpi to send and receive particle data between processors handling the force calculations of particles between boundaries. This would reduce communication overhead and allow for efficient calculations of force between particles far away from one another.

Parallel FFT: Another goal was to employ parallel FFT algorithms for calculations of the gravitational potential across the grid of particles. We expected to see an increase in speedup for long range Particle Mesh calculations as a result of this since the work required to perform these calculations would be evenly distributed between processors. However, we had trouble implementing this due to issues with the Particle Mesh.

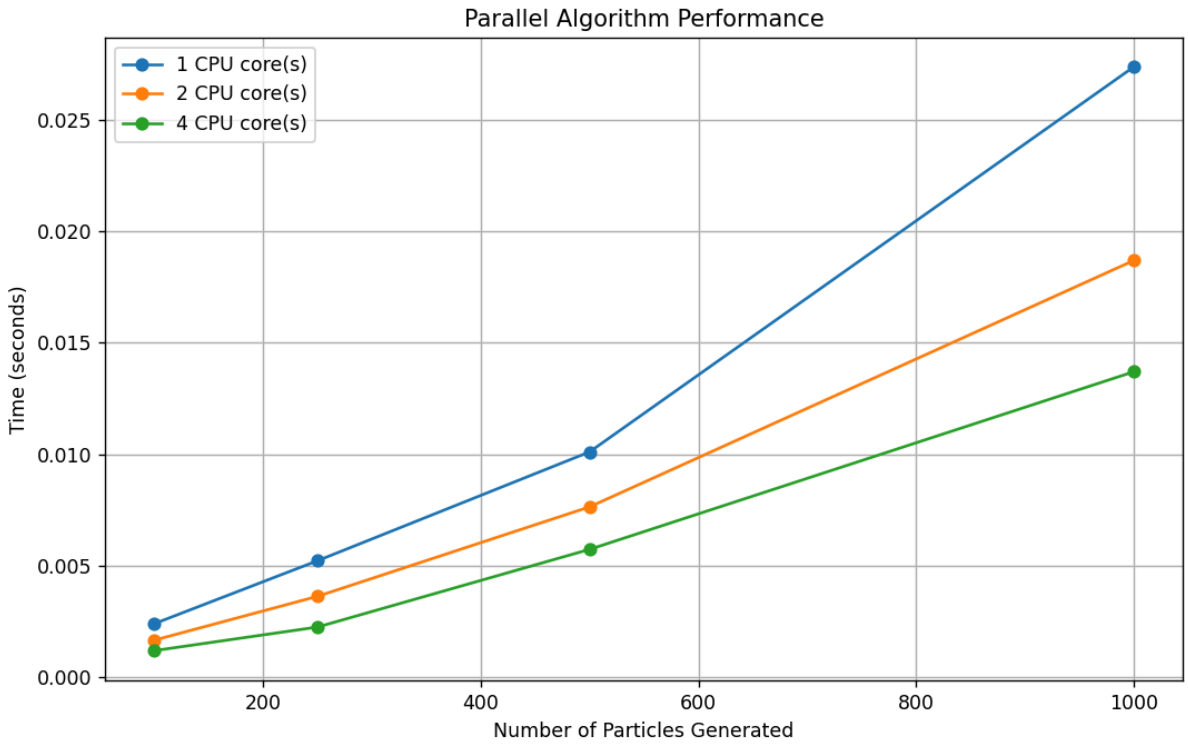
4.2 Result

4.2.1 Analysis

The Barnes-Hut algorithm has been successfully implemented and is fully functional. It effectively reduces the complexity of direct N-body calculations from O(n^2) to O(nlogn) by approximating distant particles as a single mass, thus speeding up the computation of short-range forces significantly. The TPM method, designed to efficiently estimate long-range interactions through a combination of a hierarchical tree structure and particle mesh, is currently underperforming. The main challenge lies in handling the highly sophisticated mathematics and partial differential equations integral to this method. These complexities have impeded our ability to accurately compute the potential fields over large volumes, crucial for long-distance

interaction estimation. Compared with the Naive parallel version, we observe significant reduction in the force calculation part while maintaining high accuracy levels as suggested by plot 4a. This efficiency is crucial for realistic and timely simulations.

4(a)



The parallelization of the Barnes-Hut algorithm does not achieve perfectly linear speedup due to several inherent factors related to the algorithm's structure and the nature of parallel computing. Here are the primary reasons:

1. Tree Construction Overhead:

The Barnes-Hut algorithm begins by constructing a spatial decomposition tree (quadtree in 2D, octree in 3D) where each node represents a region of space containing particles. This tree construction is a sequential process and can become a bottleneck in parallel implementations. While certain parts of the tree construction can be parallelized, the inherently hierarchical nature of tree building imposes limits on parallel efficiency.

2. Load Balancing Issues:

In the Barnes-Hut algorithm, the distribution of particles across different regions of space can be highly uneven. This leads to unbalanced workloads among different processing units (threads or cores) in a parallel system. Some processors might end up with dense clusters of

particles, requiring more computations, while others might have sparsely populated regions, resulting in idle time. Achieving optimal load balancing dynamically is challenging and impacts the linear scalability of the algorithm.

3. Communication Overhead:

Processors need to exchange information about the regions of space they are handling, particularly the mass and center of mass of the regions, to calculate forces accurately. This communication between processors can become significant, especially as the number of processors increases. The overhead from this inter-processor communication often reduces the benefits gained from parallel processing, preventing linear speedup.

4. Dependency and Synchronization:

The force calculation step in the Barnes-Hut algorithm involves walking the tree to compute gravitational forces on each particle based on the tree nodes that meet a certain criterion (usually based on a distance or "opening angle" threshold). This process has inherent dependencies because the force calculation on one particle might depend on the results of another, particularly when considering shared or bordering regions in parallel setups. These dependencies necessitate synchronization points, which can slow down execution and further deviate from linear speedup.

5. Algorithmic Complexity:

The complexity of the Barnes-Hut algorithm is O(nlogn) rather O(n^2) is the number of particles. The logarithmic term arises from the depth of the tree (as deeper trees mean more levels to traverse). In parallel environments, this complexity means that even with perfect parallelization, the speedup will inherently not be linear due to the increasing cost of deeper tree traversals with larger numbers of particles.

4.2.2 Reflection

Our algorithm features a dynamic threshold mechanism that intelligently decides the boundary between employing the Barnes-Hut approximation and the TPM method. This adaptive thresholding is based on a criterion that optimizes for both computational efficiency and accuracy, adjusting to the specific distribution and density of particles within the simulation. By combining TPM for global interactions and Barnes-Hut for local interactions, the algorithm can scale effectively to simulations involving a large number of particles, which makes it suitable for astrophysical simulations and other complex systems.The adaptive use of different methods based on the spatial distribution of particles ensures optimal computation speed without a significant compromise in accuracy. Moreover, adjustable thresholds allow the algorithm to be

finely tuned for specific simulation needs, accommodating a wide range of scenarios with varying particle densities and distribution characteristics.

4.2.3 Future Research and discussion

Continued research and development are directed towards overcoming the mathematical and computational challenges in the TPM method. Enhancing its accuracy and efficiency will provide a more balanced approach between short and long-range force calculations. Additionally, we will keep exploring additional optimization techniques for parallel processing and algorithm refinement to better handle diverse simulation scenarios and particle distributions.

The developed N-body simulation algorithm, integrating the Barnes-Hut and TPM methods enhanced by parallel computation, represents a significant step forward in the simulation of gravitational systems. While the Barnes-Hut component is effectively operational, the TPM requires further development to meet its full potential. This algorithm's advancements in computational efficiency and the ongoing efforts to overcome its current challenges are pivotal for the future of large-scale gravitational simulations.

5. Sources

Tomoaki Ishiyama, Toshiyuki Fukushige, Junichiro Makino, GreeM: Massively Parallel TreePM Code for Large Cosmological *N*-body Simulations, *Publications of the Astronomical Society of Japan*, Volume 61, Issue 6, 25 December 2009, Pages 1319–1330, https://doi.org/10.1093/pasj/61.6.1319

Bagla, Jasjeet. (2002). TreePM: A code for Cosmological N-Body Simulations. Journal of Astrophysics and Astronomy. 23. 185-196. 10.1007/BF02702282.

Contribution:

Kaiwen 50%

Alex 50%