Parallel N-Body Simulation using CUDA

Mushtaq Ahmed Shaikh
mshaikh2@ncsu.edu
NC State University
Raleigh, NC, USA

Abstract

Efficient simulation of the N-body problem is crucial for understanding complex dynamical systems, with applications ranging from astrophysics to molecular dynamics. This paper presents a high-performance CUDA-based implementation of the N-body simulation, employing both the direct-sum (DS) and Barnes-Hut (BH) algorithms. The direct-sum algorithm, characterized by a computational complexity of $O(N^2)$, is optimized for GPU acceleration using techniques such as tiling, shared memory utilization, and memory coalescence. The Barnes-Hut algorithm, with reduced complexity $O(N \log N)$, leverages hierarchical quad-tree structures, dynamic parallelism, and fixed-size array representations to minimize synchronization and optimize performance.

The experimental results demonstrate the computational efficiency of GPU-accelerated implementations. For simulations with 1 million bodies, the BH algorithm achieves a runtime of 6.32 seconds compared to 164.79 seconds on the CPU, reflecting a speedup of over 26x. For smaller scales, such as 10,000 bodies, the DS algorithm is faster on CUDA, achieving 0.165 seconds compared to 12.55 seconds sequentially, a speedup exceeding 75x. Furthermore, block size analysis reveals optimal performance at smaller block sizes (e.g., 32 threads per block), with runtime increasing significantly for larger configurations. These findings validate the scalability of the Barnes-Hut algorithm for large-scale simulations, achieving a runtime of 6.8 seconds for 10 million bodies for block size of 32.

CCS Concepts

• **Computing methodologies** → **Parallel computing**; • **Hardware** → *Graphics processors*; • **Mathematics of computing** → *Numerical algorithms*; • **Applied computing** → Astronomy; • **Software and its engineering** → High-performance computing.

Keywords

Barnes-Hut algorithm, Direct-sum algorithm, CUDA programming, GPU acceleration, Parallel computing, Dynamic parallelism, Quadtree optimization, Load balancing

1 Introduction

1.1 Problem Description

N-body simulation is a fundamental computational challenge in scientific computing, involving the prediction of dynamical interactions among a large number of particles under gravitational or other physical forces. In astrophysics, these simulations are critical for understanding complex phenomena such as galaxy formation, stellar cluster evolution, and large-scale cosmic structure development. The computational complexity of these simulations grows quadratically with the number of particles, presenting significant performance challenges for traditional sequential computing approaches.

1.2 Computational Complexity

Traditionally, N-body simulations employ two primary algorithmic strategies: direct-sum and hierarchical methods. The naive direct-sum algorithm requires $O(N^2)$ computational complexity, where N represents the number of bodies, making it computationally infeasible for large particle systems. In contrast, hierarchical algorithms like Barnes-Hut reduce computational complexity to $O(N \log N)$ by approximating distant particle interactions, enabling more efficient large-scale simulations.

1.3 Computational Parallelism

1.3.1 Force Independence and Parallelizability. The N-body gravitational problem exhibits an inherently parallel structure due to the independent force calculations between particles. Given Newton's gravitational force equation:

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

Where: - F_{ij} represents gravitational force between particles i and j - G is the gravitational constant - m_i, m_j are particle masses - r_{ij} is the distance between particles

The force calculation for each particle can be computed independently, enabling complete parallelization. Each particle's acceleration $\vec{a}_i$ can be calculated using:

$$\vec{a}_i = \sum_{j \neq i} \frac{F_{ij}}{m_i}$$

This formula demonstrates zero serial dependency, effectively achieving a parallel computational model.

1.4 Parallelism through Amdahl's Law

Amdahl's Law provides a fundamental framework for analyzing parallel computing performance, quantifying the potential speedup achieved through parallelization.

1.4.1 Theoretical Speedup Formulation. The speedup is defined as the ratio of sequential execution time to parallel execution time:

$$Speedup = \frac{T_s}{T_p}$$

Expanding this formula reveals the fundamental relationship between sequential and parallel execution:

$$Speedup = \frac{T_s}{(T_{serial} + \frac{T_{parallel}}{P})}$$

Where: - T_s is the total sequential execution time - T_p is the parallel execution time - P is the number of parallel processing units - T_{serial} represents the inherently sequential portion of the computation - $T_{parallel}$ represents the parallelizable computational segment

1.4.2 N-Body Parallelization Characterization. In N-body simulations, the serial fraction approaches zero, as force calculations for individual particles are fundamentally independent. Consequently, the speedup simplifies to:

$$Speedup \approx \frac{T_s}{T_s/P} = P$$

This theoretical analysis demonstrates that N-body simulations can achieve near-linear speedup, making them exceptionally suitable for massive parallel architectures like GPUs.

1.5 GPU Acceleration Potential

Graphics Processing Units (GPUs) offer unprecedented parallelization capabilities, making them ideal for accelerating N-body simulations. The inherently parallel nature of force calculations between particle pairs allows for significant performance improvements through massive thread-level parallelism. This work focuses on developing high-performance CUDA implementations of both direct-sum and Barnes-Hut algorithms to demonstrate the potential of GPU-based computational strategies.

1.6 Research Objectives

This work aims to:

- Implement and analyze CUDA-based direct-sum and Barnes-Hut N-body simulation algorithms
- Evaluate performance scalability across varying particle system sizes
- Compare computational efficiency between different algorithmic approaches
- Assess the impact of GPU-specific optimization techniques on simulation performance

2 Methods

The study addresses the N-body gravitational problem, which involves simulating the interactions of N particles under the influence of gravitational forces, as governed by Newton's inverse-square law. Each particle is defined by its initial position ($\mathbf{P}_i = (x_i, y_i, z_i)$), velocity ($\mathbf{V}_i = (v_{ix}, v_{iy}, v_{iz})$), and mass ($m_i$). The objective is to compute the trajectories of these particles over time by iteratively solving for their accelerations ($\mathbf{A}_i$) and updating their velocities and positions.

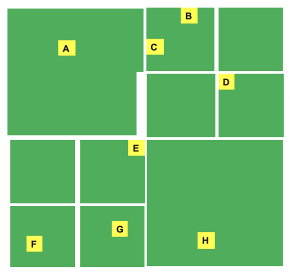
Two distinct computational approaches were implemented for this purpose:

1. Direct Sum Algorithm: A naive approach with $O(N^2)$ computational complexity, where forces are calculated explicitly for all N-body pairs. CUDA optimizations for this method include:
- **Tiling**: Optimizes data reuse and ensures efficient memory access patterns.
- **Shared Memory Utilization**: Minimizes global memory accesses by caching data locally, improving performance.
- **Memory Coalescing**: Ensures aligned memory accesses for threads, reducing latency and enhancing throughput.

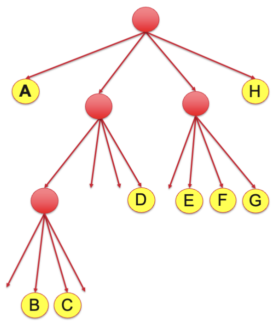
Algorithm 1 Naive Direct Sum Algorithm for N-body Simulation

```
1: for each body $i$ in the system do
2:     for each body $j \neq i$ do
3:         Compute $F_{ij}$ using Newton's law
4:         Update acceleration $\mathbf{a}_i$ for body $i$
5:     end for
6:     Compute velocity $\mathbf{v}_i$ for body $i$
7:     Update position $\mathbf{p}_i$ for body $i$
8: end for
```

2. Barnes-Hut Algorithm: An approximation algorithm with $O(N \log N)$ complexity. It groups distant bodies and approximates their collective influence as a single "large body." [2] The Barnes-Hut method involves:
- **Quad-Tree Construction**: Organizing the simulation space hierarchically into a tree structure, where each node represents a region of space containing bodies.
- **Force Calculation**: Starting from the root of the quad-tree, the algorithm recursively evaluates forces based on the ratio s/d, where s is the node width and d is the distance to the node's center of mass. Nodes are explored further only if they do not meet the approximation threshold.



(a) Bodies distributed across space



(b) Quad-Tree construction

Figure 1: Bodies distributed across space are divided into grids. Each grid is broken down into sub-grids until there is a single body left in the grid or the recursion depth limit hits. The Quad-Tree is constructed with the center of the grid as root node and nodes are constructed in-order of NW NE SE SW

3 Implementation Details

The CUDA implementation was designed to maximize parallelism and optimize memory access for both algorithms.

3.1 Direct Sum CUDA Implementation

- Threads are assigned to compute the interactions of a subset of particles, with P threads performing P^2 computations.

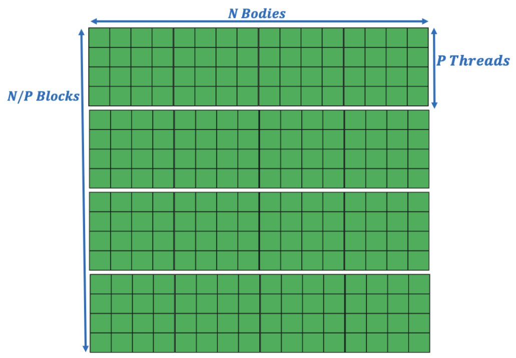


Figure 2: Tiling strategy for Direct Sum CUDA implementation [1]

- **Tiling** is utilized to partition data into manageable chunks, enabling data reuse within the shared memory of each thread block.
- The use of shared memory minimizes expensive global memory accesses, while memory coalescing ensures efficient memory transfer to the GPU.

3.2 Barnes-Hut CUDA Implementation

The Barnes-Hut algorithm is implemented using CUDA to hierarchically divide the simulation space into quadrants, constructing a quadtree in parallel. Each CUDA thread block processes a node, subdividing it further as necessary. Key features of the implementation include:

- **Dynamic Parallelism**: Threads dynamically launch child threads to process sub-nodes, enabling recursive quadtree construction.
- **Fixed-Size Array Quadtree**: Nodes are indexed for traversal, avoiding pointers and synchronization overhead, thus optimizing memory access.
- **Recursive Force Calculations**: Tree traversal balances accuracy and performance, approximating distant nodes to reduce computational costs.

The algorithm is divided into four CUDA kernels:

Kernel 1: Reset Quadtree Nodes

Each quadtree node is reset in parallel, with one thread assigned per node. This step initializes the structure for subsequent kernels.

Kernel 2: Compute Bounding Box

The root node's bounding box, encompassing the entire 2D simulation space, is computed in parallel. The bodies are split among thread blocks, where each block calculates the local bounds using parallel reduction. Shared memory ensures efficient data access, while atomic operations prevent race conditions during updates to the global bounds.

Kernel 3: Construct Quadtree

A top-down hierarchical approach constructs the quadtree without locks or synchronization. At depth 0, a single thread block divides the entire space into quadrants. For each quadrant, child blocks are dynamically launched to process further subdivisions. Each thread block calculates its node's center of mass via parallel reduction, ensuring computational efficiency. The process includes grouping bodies into quadrants and updating child nodes' indices, leveraging atomic operations for accurate count and parallel scan for offset calculation.

Kernel 4: Compute Force

Force computation employs depth-first traversal of the quadtree, leveraging recursive calls to minimize memory overhead compared to a queue-based iterative approach. Each body interacts with others based on proximity, selectively approximating distant nodes to balance accuracy with performance.

Algorithm 2 Barnes-Hut CUDA Implementation

```
1: for each timestep do
2:     Kernel 1: Reset Quadtree
3:     Kernel 2: Compute Bounding Box
4:     Kernel 3: Construct Quadtree
5:     Kernel 4: Compute Force
6: end for
```

4 Implementation Details

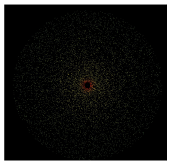
This section outlines the computational methods, tools, and configurations used to evaluate the performance of the N-body simulations. The experiments compare the Barnes-Hut (BH) and Direct Summation (DS) algorithms, implemented using both GPU (CUDA) and CPU-based sequential approaches.

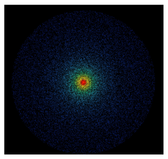
4.0.1 Hardware Configuration. The experiments were conducted on a high-performance computing machine with the following specifications:

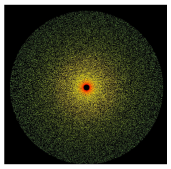
- **GPU**: NVIDIA GeForce RTX 2080 Ti
- **CUDA Version**: 12.4
- **Driver Version**: 550.120
- **GPU Memory**: 11GB GDDR6
- **Maximum Power Usage**: 260W

This GPU architecture was selected for its ability to handle computationally intensive tasks, particularly through parallel processing and high memory bandwidth. The CPU used for sequential simulations is part of the same system, ensuring consistency in baseline comparisons.

4.0.2 Simulation Images. Below are the visualizations of the N-body simulations for different numbers of bodies:


(a) N = 10K bodies.


(b) N = 50K bodies.


(c) N = 100K bodies.

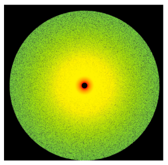

(d) N = 1M bodies.

Figure 3: Simulations for various numbers of bodies: 10K, 50K, 100K, and 1M.

4.0.3 Simulation Time vs. N-Bodies (Up to 10K). The simulation time for small-scale simulations (up to 10K bodies) demonstrates the relative performance of the four methods: Barnes-Hut (BH) CUDA, Direct Summation (DS) CUDA, BH Sequential (Seq), and DS Sequential.

N-Bodies	BH Cuda	DS Cuda	BH Seq	DS Seq
1000	0.0453	0.0260	0.0688	0.1382
2000	0.0523	0.0162	0.1377	0.5267
5000	0.0758	0.0562	0.4318	3.1617
10000	0.1235	0.1660	0.8645	12.5522

Table 1: Simulation time (in seconds) for up to 10K bodies.

From Table 1, we observe that GPU-based implementations (BH Cuda and DS Cuda) significantly outperform their sequential counterparts. The DS CUDA implementation shows the lowest simulation time for small numbers of bodies, indicating its efficiency in handling particle interactions without the overhead of tree construction. Conversely, the BH CUDA method becomes more competitive as the number of bodies increases, due to its ability to efficiently approximate long-range interactions.

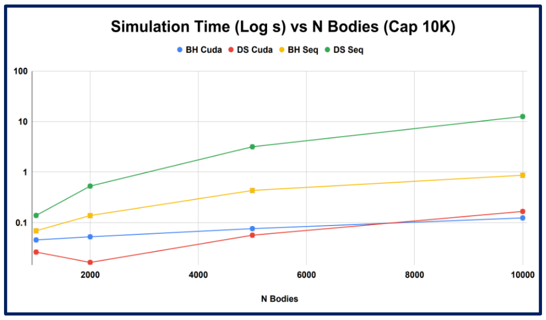


Figure 4: Simulation Time (Log scale) vs. N-Bodies (Up to 10K).

Figure 4 illustrates the rapid scaling of sequential methods compared to CUDA-based methods. Notably, the DS Sequential method becomes extensively slow even for 10K bodies.

4.0.4 Simulation Time vs. N-Bodies (10K to 1M). As the number of bodies increases, the performance gap between GPU and CPU implementations widens significantly. Table 2 provides simulation times for up to 1 million bodies.

N-Bodies	BH Cuda	DS Cuda	BH Seq	DS Seq
10000	0.1235	0.1660	0.8645	12.5522
20000	0.2154	0.5144	1.9568	50.9201
50000	0.4612	2.7261	5.2275	328.4350
100000	0.8145	10.4594	12.6387	1283.35
1000000	6.3171	1028.94	164.79	–

Table 2: Simulation time (in seconds) for 10K to 1M bodies.

The DS Sequential method is unable to handle simulations beyond 100K bodies due to excessive computational overhead. The BH CUDA implementation shows remarkable scalability, with simulation times increasing sub-linearly relative to the number of bodies. DS CUDA, on the other hand, becomes inefficient for large-scale simulations due to its quadratic complexity.

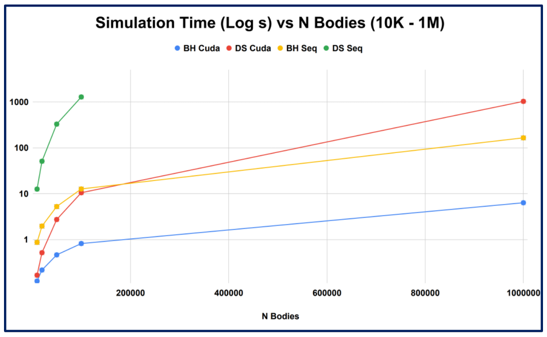


Figure 5: Simulation Time (Log scale) vs. N-Bodies (10K to 1M). The Barnes-Hut CUDA implementation maintains scalability, whereas the DS CUDA implementation exhibits exponential growth for larger simulations.

4.0.5 Block Size Performance. The impact of block size on simulation time was analyzed for 5M and 10M bodies. Table 3 summarizes the results.

Block Size	5M Bodies	10M Bodies
32	4.7361	6.8031
64	8.3364	12.8558
128	23.8591	42.9403
256	23.9041	43.6066
512	24.0711	43.5504

Table 3: Performance comparison for varying block sizes.

For smaller block sizes (32 and 64), the GPU achieves better performance due to reduced thread divergence. However, for larger block sizes (128 and above), performance stabilizes as thread scheduling and memory utilization reach their limits.

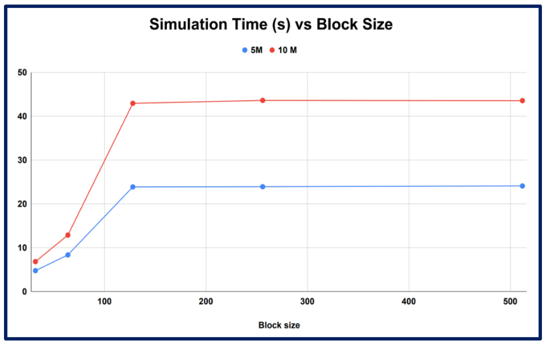


Figure 6: Simulation Time vs. Block Size. The results indicate that block sizes of 32 and 64 offer optimal performance for our GPU configuration.

5 Conclusion

This study evaluates the performance of the Barnes-Hut (BH) and Direct Summation (DS) algorithms for N-body simulations, with a focus on GPU acceleration using CUDA. The GPU-accelerated Barnes-Hut algorithm demonstrates significant performance improvements, achieving faster computation times for large-scale simulations due to optimizations like memory coalescing and dynamic parallelism. In contrast, the Direct Summation method, though accurate, remains computationally expensive and benefits less from GPU acceleration.

References

[1] Lars Nyland, Mark Harris, and Jan Prins. 2007. Chapter 31. fast n-body simulation with cuda. https://developer.nvidia.com/gpugems/gpugems3/part-v-physics-simulation/chapter-31-fast-n-body-simulation-cuda

[2] Wikipedia. 2023. Barnes–Hut simulation — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Barnes%E2%80%93Hut%20simulation&oldid=1092207998 Online; accessed 26-February-2023.