Accelerating Barnes-Hut N-Body Simulations Using CUDA on RTX 4090 GPUs

John Barnes VI
Department of Computer Science
University of Houston
Houston, United States
jabarne6@CougarNet.UH.EDU

Yash Patel
Department of Computer Science
University of Houston
Houston, United States
yspatel2@CougarNet.UH.EDU

Hitarth Thanki
Department of Computer Science
University of Houston
Houston, United States
hmthanki@CougarNet.UH.EDU

***Abstract*—This paper presents a GPU-accelerated implementation of the Barnes-Hut algorithm for 2D N-body simulations using NVIDIA's CUDA framework. While achieving significant performance improvements over the naive $O(N^2)$ approach, our implementation faces bottlenecks due to CPU-based quadtree construction. We discuss challenges in parallelizing spatial indexing structures on GPUs, analyze performance characteristics up to 1 million particles, and propose future directions using advanced parallel primitives inspired by recent research in GPU-accelerated spatial indexing.**

***Index Terms*—N-body simulation, Barnes-Hut, CUDA, quadtree, GPU acceleration**

I. Introduction

N-body simulations are fundamental computational tools in astrophysics, molecular dynamics, and computer graphics. The naive approach to calculating gravitational interactions exhibits $O(N^2)$ complexity, becoming computationally prohibitive for $N > 10^4$ particles. The Barnes-Hut algorithm reduces this complexity to $O(N \log N)$ through spatial hierarchy and approximation. Modern GPUs like the RTX 4090 offer massive parallelism but require careful algorithm design to fully exploit their capabilities.

II. Background

A. N-Body Problem Fundamentals

The classical N-body problem involves simulating the motion of particles interacting through pairwise gravitational forces. For particle i, the net force is calculated as:

$$F_i = G \sum_{\substack{j=1 \\ j \neq i}}^{N} \frac{m_i m_j}{\|r_{ij}\|^3} r_{ij} \tag{1}$$

where r_{ij} is the distance vector between particles i and j, and m represents particle masses.

B. Barnes-Hut Algorithm

The Barnes-Hut algorithm approximates long-range forces using spatial hierarchy:

1) Spatial decomposition into recursively subdivided quadrants
2) Multipole acceptance criterion (MAC): $\frac{s}{d} < \theta$
3) Aggregate force calculations for distant cell clusters

where s is cell size, d is distance to cell's center of mass, and θ is the approximation threshold (typically 0.5).

C. Morton Codes and Spatial Indexing

Morton codes (Z-order curves) provide an efficient spatial indexing mechanism by interleaving coordinate bits to preserve multidimensional locality in linear ordering. For 2D coordinates (x, y), the Morton index Z is computed as:

$$Z(x, y) = \bigoplus_{k=0}^{b-1} (x_k \ll (2k+1)) \oplus (y_k \ll 2k) \tag{2}$$

where $\oplus$ denotes bitwise OR, $\ll$ bit shifting, and x_k/y_k the k-th bit of normalized coordinates. Each pair of bits in Z encodes quadrant membership:

- **00**: Southwest (SW) quadrant
- **01**: Southeast (SE) quadrant
- **10**: Northwest (NW) quadrant
- **11**: Northeast (NE) quadrant

This encoding enables efficient spatial hierarchy construction through bitwise operations. At each quadtree level, two bits from the Morton code determine quadrant membership, with successive bits refining spatial resolution. The Z-ordering property ensures that:

- Spatially proximate points have similar Morton codes
- Sorting by Z-order clusters particles by spatial locality
- Hierarchical relationships become decipherable through bitmask operations

Our implementation maps normalized particle coordinates to 32-bit Morton indices, enabling GPU-friendly spatial sorting via parallel radix sort. Each quadtree level examines successive bit pairs to determine quadrant subdivisions, with SW=00, SE=01, NW=10, NE=11 following standard spatial quadrant numbering conventions.

III. Implementation

A. System Architecture

Our CUDA 12.2 implementation employs a three-stage pipeline: First, particle positions are projected onto Z-order Morton codes using parallel transformations. The Morton-sorted data then drives quadtree construction on the host CPU, which remains necessary due to the recursive nature of spatial partitioning. Finally, the force calculation kernel traverses the tree structure while maintaining 512-thread blocks for optimal GPU utilization.

Key architectural components include:

- Morton code generation through bitwise interleaving of spatial coordinates
- Thrust-based parallel primitives for boundary detection and sorting
- Zero-copy OpenGL buffer mapping for real-time visualization
- Asynchronous memory transfers overlapping with kernel execution

B. GPU Acceleration Strategy

The implementation optimizes GPU utilization through:

- Warp-level coalesced memory access patterns
- Shared memory caching of frequently accessed tree nodes
- Block-level parallel reduction for multipole moment calculations
- Persistent kernel design minimizing launch overhead

IV. Development Challenges

A. Quadtree Construction Bottleneck

Our initial GPU quadtree implementation encountered three fundamental constraints of massively parallel architectures:

- **Recursive-Parallel Duality**: Tree construction's inherently sequential nature conflicted with GPU's SIMT execution model
- **Memory Fragmentation**: Dynamic node allocation caused irregular memory access patterns
- **Control Flow Divergence**: Threads processing different tree branches exhibited variance in execution paths

The current CPU-based compromise allows for stability up to 1M particles but reveals critical scaling limitations for non-parallelized tree-construction:

- Host-side tree construction takes up a larger portion of runtime as n increases
- PCIe 4.0 bandwidth saturation from memory transfers as n increases

V. Results and Analysis

TABLE I
Performance Characteristics (RTX 4090)

Metric	Naive	Barnes-Hut
Breakdown Point	100k	1M
Complexity	$O(N^2)$	$O(N \log N)$

Performance analysis reveals three critical operational thresholds:

- **Naive Method Limit**: Direct $O(N^2)$ calculations become impractical beyond 100k particles due to quadratic time growth
- **Barnes-Hut Scalability**: Maintains real-time performance up to 1M particles through hierarchical approximation
- **Critical Threshold**: 800k particles where quadtree construction becomes dominant and slow degradation begins

The Barnes-Hut implementation with CPU quadtree construction demonstrates an order of magnitude improvement over the naive approach at 1M particles, though host-side tree construction remains a bottleneck. This confirms the theoretical $O(N \log N)$ complexity while exposing memory transfer bottlenecks between CPU and GPU components.

VI. Future Work

A. Bottom-Up Construction

Recent advances in [1] suggest bottom-up methods could eliminate host bottlenecks through:

- Single-pass Morton sorting at maximum spatial resolution
- Parallel quadrant aggregation using prefix scans
- Structure-of-Arrays (SoA) memory layout minimizing cache misses
- Atomic-free node counting via ballot/shuffle intrinsics

B. Top-Down Optimization

The referenced top-down approach from [1] offers alternative optimizations:

- Level-ordered parallel partitioning with bitmask operations
- Warp-specialized bucket sorting using CUDA 12's coalesced groups
- Hybrid CPU/GPU construction for upper tree levels
- Compressed sparse node representation

Both strategies must address:

- Memory overhead for intermediate structures
- Load balancing across 72 streaming multiprocessors
- L2 cache contention during global reductions

VII. Conclusion

Our implementation demonstrates effective Barnes-Hut acceleration on modern GPUs while exposing fundamental challenges in parallel spatial indexing. The increasingly dominant quadtree construction latency at 1M particles highlights the critical need for GPU-native tree building strategies. Emerging bottom-up and top-down approaches from recent literature provide viable pathways toward real-time billion-particle simulations, contingent on solving memory access pattern optimization and load balancing challenges inherent to hierarchical spatial decomposition.

References

[1] J. Zhang and L. Gruenwald, "Efficient Quadtree Construction for Indexing Large-Scale Point Data on GPUs: Bottom-Up vs. Top-Down," 2023.
[2] NVIDIA, "CUDA C Programming Guide v12.2," 2023.
[3] NVIDIA, "Thrust Parallel Algorithms Library Documentation," 2023.