GPU-Optimized N-Body Simulation Using Barnes-Hut Algorithm

Team members: Mandar Dhamale(mandardhamale@usf.edu), James Hindmarch(hindmarch@usf.edu)

1. Introduction

1.1 Problem Statement

N-body simulations calculate how multiple objects (planets, stars, particles) move under mutual gravitational attraction. The challenge is that each object affects every other object, making a direct approach extremely slow for large numbers of objects.

1.2 Solution Approach

This project implements the **Barnes-Hut algorithm**, which groups distant objects together to reduce calculations. The implementation is optimized for **NVIDIA GPUs** using CUDA for parallel processing.

1.3 Key Achievement

Our implementation reduces calculation time from **$O(N^2)$** (direct method) to **O(N log N)** while maintaining acceptable accuracy, enabling the simulation of thousands of particles in real-time.

2. Algorithm Design

2.1 The Barnes-Hut Approximation

- **Simple Analogy:** When viewing a distant forest, you don't see individual trees, you see a green mass. Similarly, distant particle groups are treated as single masses.
- **Key Concept:** The *opening angle* (theta = 0.7) determines when to use approximations:
  - **Small angle:** More accurate but slower.
  - **Large angle:** Faster but less accurate.

2.2 Data Structures

2.2.1 Particle Data (GPU-Optimized)

Each particle stores:

- **Position:** (x, y, z coordinates)
- **Velocity:** (speed in each direction)
- **Acceleration:** (current change in velocity)
- **Mass:** (how heavy it is)

Optimization: Data is packed tightly (16-byte aligned) for fast GPU memory access.

2.2.2 Octree Structure

Think of this as a **3D space organizer**:

1. Space is divided into 8 *octants* (like sub-boxes).
2. Each octant can be subdivided further.

3. Particles in distant octants are grouped together.

Visual Analogy:

Initial space: [ ]
First division: [ ][ ][ ][ ]
...
Leaf contains: [*][*][*] (individual particles)

2.3 Calculation Pipeline

1. **Step 1: Build Octree**
   - Sort particles into spatial boxes.
   - Calculate group masses and centers.
2. **Step 2: Calculate Forces (Hybrid Approach)**
   - *Nearby particles:* Exact calculation (shared memory).
   - *Distant groups:* Approximate as single mass (tree).
3. **Step 3: Update Positions**
   - Move particles based on calculated forces.
   - Update velocities.

3. GPU Optimizations

3.1 Algorithmic Optimization (O(N log N))

- **Problem:** The naive "all-pairs" approach requires checking everybody against every other body (N squared interactions), which effectively freezes the simulation at high particle counts.
- **Solution:** Implemented the Barnes-Hut algorithm using an Octree data structure. The kernel utilizes the **Multipole Acceptance Criterion (MAC)**: if a distant cluster of particles (a tree node) is far enough away (size / distance < theta), it is treated as a single super-particle.
- **Benefit:** Reduces computational complexity from quadratic O(N^2) to quasilinear O(N log N), reducing interactions from approx. 100 million to approx. 5-10 million per frame for 10,000 bodies.

3.2 Memory Layout Optimization (Coalescing)

- **Problem:** Standard GPU memory transactions occur in 128-byte chunks. A standard particle struct is often 40-44 bytes, which causes "unaligned" memory access, requiring the GPU to issue multiple transactions to read a single value.
- **Solution:** Implemented `struct alignas(16)` for the `Body` and `OctreeNode` structures.
- **Benefit:** Ensures **Memory Coalescing**. The GPU can read data for multiple threads in single, wide transactions, significantly maximizing global memory bandwidth.

3.3 Iterative Tree Traversal (Stack-Based)

- **Problem:** The standard Barnes-Hut algorithm relies on **recursion** to traverse the tree. GPUs have very small stack memory per thread, and deep recursion causes "register spilling" (dumping fast memory to slow global memory) or kernel crashes.
- **Solution:** Replaced recursion with an **iterative while loop** managing a custom, fixed-size stack (`int stack[64]`) within the kernel registers.

- **Benefit:** Eliminates function call overhead and prevents stack overflow, allowing efficient traversal of deep trees on the GPU hardware.

3.4 Hardware Intrinsic Math

- **Problem:** Standard mathematical operations like square roots and division are computationally expensive (taking many clock cycles).
- **Solution:** Utilized CUDA hardware intrinsics, specifically `rsqrtf(x)` (Reciprocal Square Root).
- **Benefit:** Maps directly to a single hardware instruction on NVIDIA GPUs, calculating 1/sqrt(x) much faster than the standard division sequence.

4. Test Cases for Evaluation

4.1 Visual Test: Stable Orbits

- **Purpose:** Verify basic correctness, particles should orbit without flying apart.
- **How to Test:**

```
# 1. Compile the code
nvcc optimized_barnes_hut.cu -o optimized_simulation -O3 -arch=sm_70

# 2. Run the simulation
./optimized_simulation

# 3. Generate visualizations
python visualize_simple.py
```

- **Expected Output Files:**
  - simulation_output.txt - Raw particle positions
  - barnes_hut_results.png - Three snapshots (beginning, middle, end)
  - barnes_hut_animation.gif - Full animation
- **Success Criteria:** Animation shows coherent orbital motion (disk-like patterns, no escaping particles).

4.2 Performance Scaling Test

- **Purpose:** Verify the algorithm scales efficiently.
- **Test Procedure:** Edit main() in optimized_barnes_hut.cu:

```
// Change this line to test different sizes
const int N_BODIES = 512;  // Try: 512, 1024, 2048, 4096
```

- **Expected Results:**

N_Bodies	Time (ms)	Time/N Ratio	Evaluation
512	~250	1.0x	Baseline
1024	~500	~2.0x	Good: O(N log N)

2048	~1000	~4.0x	Good: O(N log N)
4096	~2200	~8.8x	Good: slightly better than O(N^2)

4.3 Accuracy Validation Test

- **Purpose:** Ensure approximations don't break the simulation.
- **Procedure:** Modify code to bypass tree traversal (Direct Calculation) for small N (256) and compare outputs.
- **Acceptable Difference:** Positions should differ by < 5% after 100 steps.

6. Expected Results

For Default Settings (2048 particles, 100 steps):

- **Execution Time:** 500-1000 ms
- **Visual Output:** Stable orbital disk with spiral patterns
- **GPU Memory:** ~700 MB
- **Performance:** 50-100× faster than equivalent CPU code

Visual Guide for Graders

Good Simulation	Bad Simulation
Stable Orbits: Particles in disk shape	**Chaos:** Particles scattered everywhere
Smooth: Coherent orbital motion	**Random:** Chaotic motion
Contained: No escapees	**Broken:** Many particles at edges

7. Troubleshooting Guide

Issue: Compilation error about architecture

Configuration of CUDA on development machine:

```
[~]
base  mandar-dhamale  nvidia-smi
Thu Dec  4 23:24:12 2025
+-----------------------------------------------------------------------------------------+
| NVIDIA-SMI 580.95.05              Driver Version: 580.95.05      CUDA Version: 13.0     |
+-----------------------------------------+------------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute M. |
|                                         |                        |               MIG M. |
|=========================================+========================+======================|
|   0  NVIDIA GeForce RTX 3060 ...    Off |   00000000:01:00.0  On |                  N/A |
| N/A   40C    P8             16W /   80W |      64MiB /   6144MiB |     10%      Default |
|                                         |                        |                  N/A |
+-----------------------------------------+------------------------+----------------------+

+-----------------------------------------------------------------------------------------+
| Processes:                                                                              |
|  GPU   GI   CI              PID   Type   Process name                        GPU Memory |
|        ID   ID                                                               Usage      |
|=========================================================================================|
|    0   N/A  N/A            4246      G   /usr/lib/xorg/Xorg                       45MiB |
+-----------------------------------------------------------------------------------------+
```

- **Fix:** Check your GPU compute capability.
  nvcc optimized_barnes_hut.cu -o optimized_simulation -O3 -arch=sm_75
  # Try values: sm_50, sm_60, sm_70, sm_75, sm_80

Issue: Python visualization fails

- **Fix:** Install required packages.
  pip install numpy matplotlib

Issue: Simulation runs slowly

- **Fix:** Reduce particle count in main().
  const int N_BODIES = 1024; // Instead of 2048

Issue: No output files

- **Fix:** Check write permissions.
  chmod 755 optimized_simulation

8. Conclusion

This project successfully implements a GPU-accelerated Barnes-Hut algorithm that:

1. Correctly simulates **gravitational interactions**.
2. Scales efficiently with particle count.
3. Leverages **GPU parallelism** for significant speedup.
4. Produces visually verifiable results.

The test cases provide straightforward ways to validate correctness and performance without requiring physics expertise. The implementation demonstrates the practical application of parallel computing principles to solve computationally intensive problems efficiently.

Key Innovation: The hybrid approach combining direct calculation for nearby particles with tree approximation for distant ones achieves an optimal balance between accuracy and performance, making large-scale N-body simulation accessible on consumer hardware.