# Profiling Report

N-Body Simulation with Barnes Hut Optimisation

Name: Sanshrey Roll Number: CS23B2014

## Profiling Report: N-Body Simulation (Quantifiable Metrics)

This report provides specific, quantifiable data on the simulation's performance, identifying exactly where time is spent and which code paths are executed most frequently.

### 1. Hot Functions (Time & Calls)

The following functions consume the most CPU time. **Optimization efforts should focus strictly on the top 2 functions.**

| Function Name | % of Time | Self Time (s) | Call Count (Gprof) | Call Count (Gcov)* |
|---|---|---|---|---|
| **`compute_force_on_particle`** | **80.68%** | **15.95s** | 510,000 | **20,700,160** |
| `_init` | 9.71% | 1.92s | - | - |
| `OctreeNode::can_approximate` | 5.16% | 1.02s | **1,058,930,495** | 1,058,930,495 |
| `std::_Function_handler...` | 3.95% | 0.78s | 20,000 | - |

> ***Note****: Gprof counts often miss recursive calls. Gcov provides the accurate execution count for the function body.*

**Insight**: - `OctreeNode::can_approximate` is called **over 1 billion times**. Even nanosecond-level optimizations here (like removing a branch or using reciprocal multiplication) will have a massive cumulative effect. - `compute_force_on_particle` accounts for ~**16 seconds** of the ~20-second runtime.

### 2. Hot Paths (Execution Frequency - Gcov)

The "Hot Path" is the specific sequence of lines executed most frequently.

**Trace: `BarnesHutSolver::compute_force_on_particle`**

| Code Block / Event | Line No. | Execution Count | Impact |
|---|---|---|---|
| **Recursive Tree Traversal** | `barnes_hut.cpp:52` | **23,275,305** | **CRITICAL**: The loop iterating over 8 children is the hottest loop in the program. |
| **Recursive Function Call** | `barnes_hut.cpp:53` | **20,689,160** | Each iteration makes a recursive call, adding stack overhead. |
| **Node Type Check (Empty/Leaf)** | `barnes_hut.cpp:19` | 20,700,160 | Evaluated for every visited node. |

| Code Block / Event | Line No. | Execution Count | Impact |
|---|---|---|---|
| **Leaf processing (Direct Force)** | `barnes_hut.cpp:26` | 5,015,153 | Actual force calculation happens ~5 million times. |
| **Internal Node (MAC Check)** | `barnes_hut.cpp:39` | 4,195,196 | The decision to approximate or recurse is made ~4.2 million times. |
| **Approximation Used** | `barnes_hut.cpp:43` | 1,609,051 | We successfully approximate ~1.6 million times (38% of internal node checks). |

**Quantifiable Insight**: - For every 1 successful approximation (saving work), we traverse ~13 tree nodes (20.7M visits / 1.6M approximations). - The "Hot Path" is: `Enter Function` -> `Check Node Type` -> `Loop 8 Children` -> `Recurse`.

## 3. Hardware Performance Metrics (Likwid)

These metrics quantify *how* the processor executes the hot paths.

| Metric | Value | Interpretation |
|---|---|---|
| **Total Runtime** | 38.40 s | Wall-clock time for the parallel region. |
| **DP MFLOP/s** | **241.79** | **Low**. The CPU is performing floating-point ops at a fraction of its peak theoretical speed (GFLOPS range). This confirms the code is **not compute-bound**. |
| **CPI (Cycles Per Instruction)** | **1.14** | **Mediocre**. In a high-performance compute kernel, we expect CPI < 0.5. A value > 1.0 means the CPU is stalling, likely waiting for data. |
| **Clock Frequency** | 2.0 GHz | The CPU is running at ~2.0 GHz on average (Base is 3.3 GHz). |
| **Retired Instructions** | 66.37 Billion | Total work done. |

**Conclusion**: The low MFLOP/s combined with high call counts and pointer-chasing logic (tree traversal) confirms the code is **Latency Bound**. The CPU spends most cycles waiting to fetch `OctreeNode` data from memory rather than computing forces.