Parallelizing 2D Collision Detection

Experimenting on 2D collision detection with CUDA via fixed-grid structures and adaptive spatial partitioning (Quadtree) to handle large-scale, imbalanced particle distributions efficiently.

15-418/618, Fall 2024 Parallel Computer Architecture and Programming   Authors: Kewei Han, Jiya Zhang

01. Introduction

Problem: Simulating collisions among thousands of particles in 2D is computationally expensive. A naive $O(n^2)$ check for collisions each frame limits scalability and reduces rendering frame rates.

Goal: Use a fixed spatial grid or a Quad-tree to reduce the number of collision checks and improve collision computation performance. Apply CUDA-based parallelization to accelerate the collision resolution step.

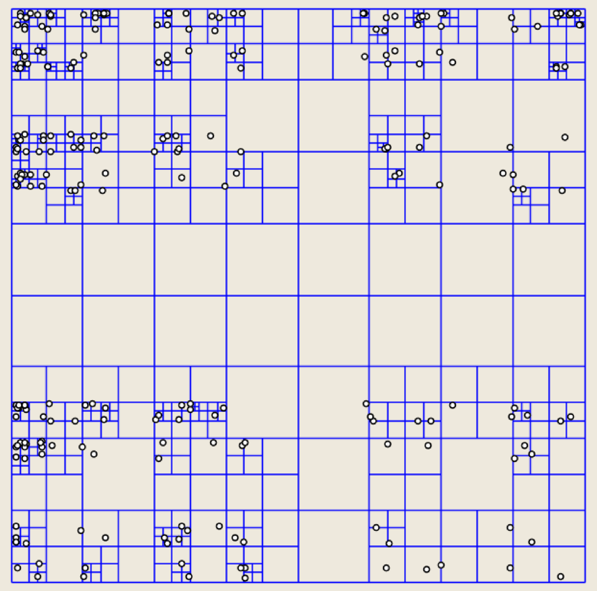


Figure 1: Quadtree

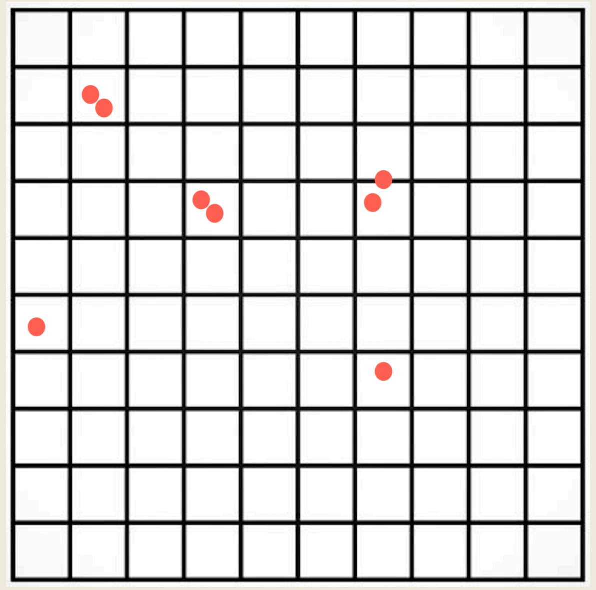
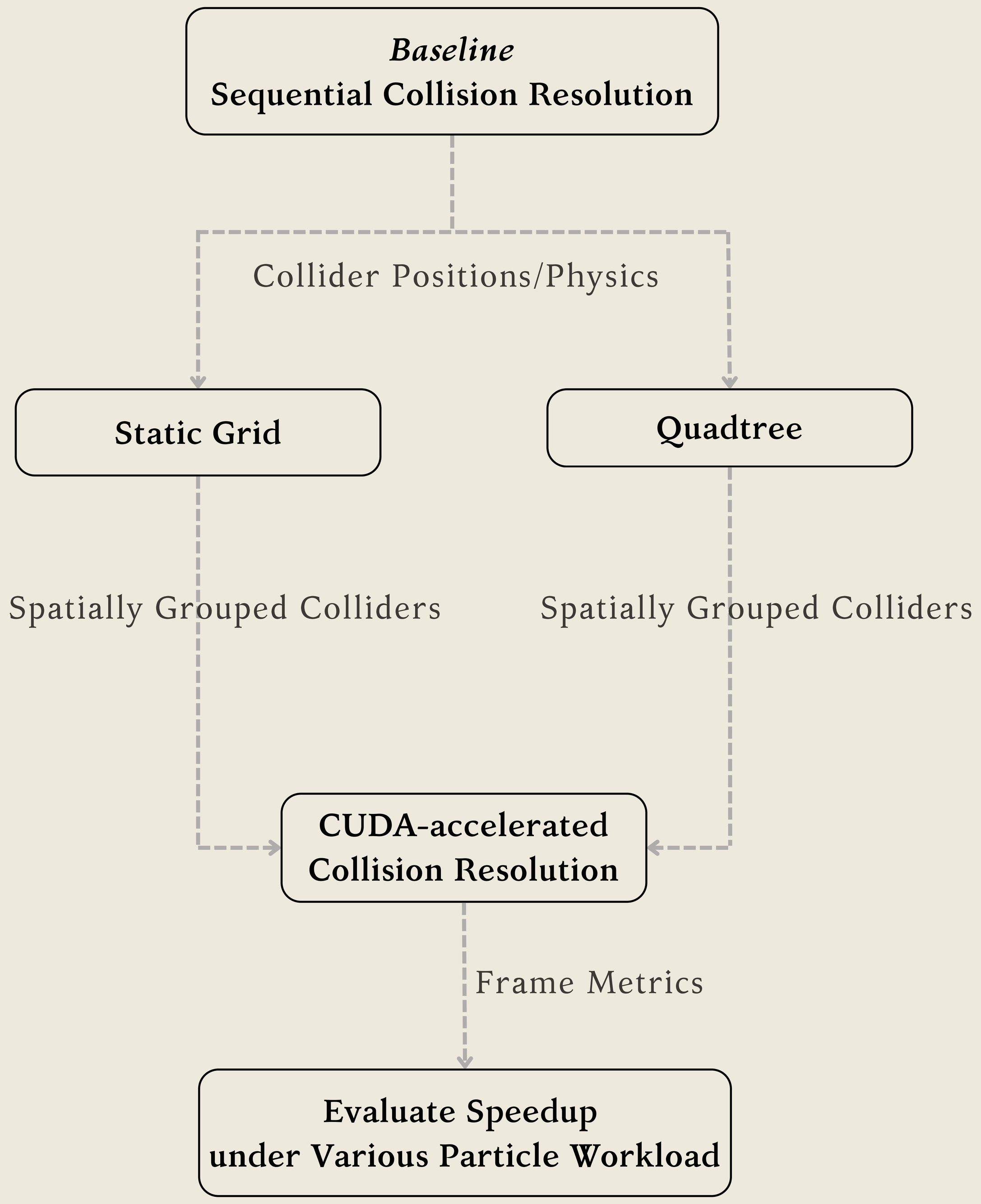


Figure 2: Static Grid

02. Optimization Strategy & Algorithm



03. Experimental Results

(1) Performance Comparison Based on Number of Entities under Uniform Distribution

- Small & sparse workloads: **Quadtree** sequential implementation is highly efficient due to its ability to adaptively partition space with minimal overhead; CUDA-based methods have higher overhead.
- Large workload: **CUDA** implementation under both Static Grid and Quadtree structure outperforms the other methods as it maximizes parallelism and scalability.

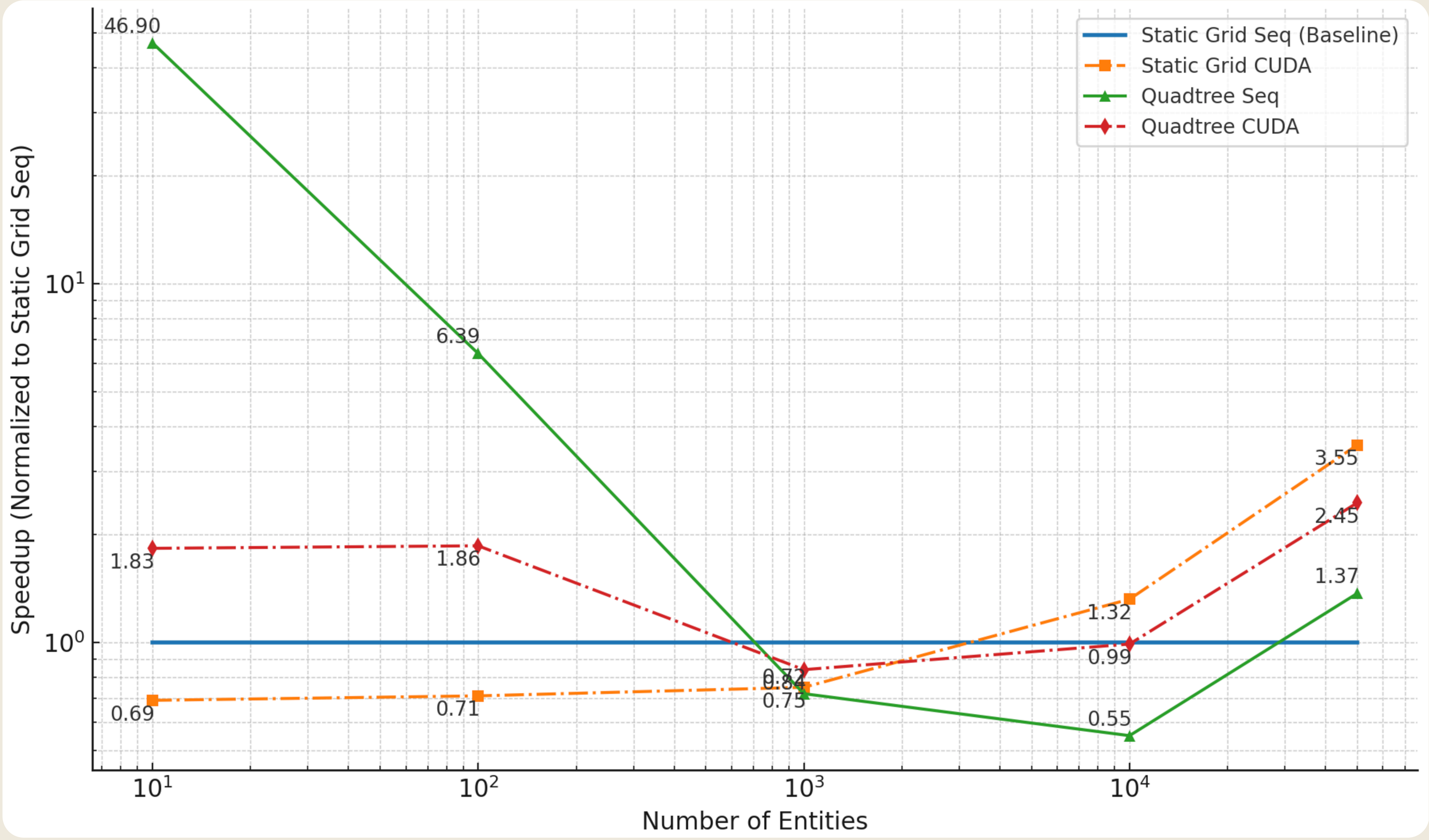


Figure 3: Performance Evaluation Based on Entity Count

(2) CUDA Implementations Performance Comparison on Imbalanced and Balanced Distribution (1700 Entities)

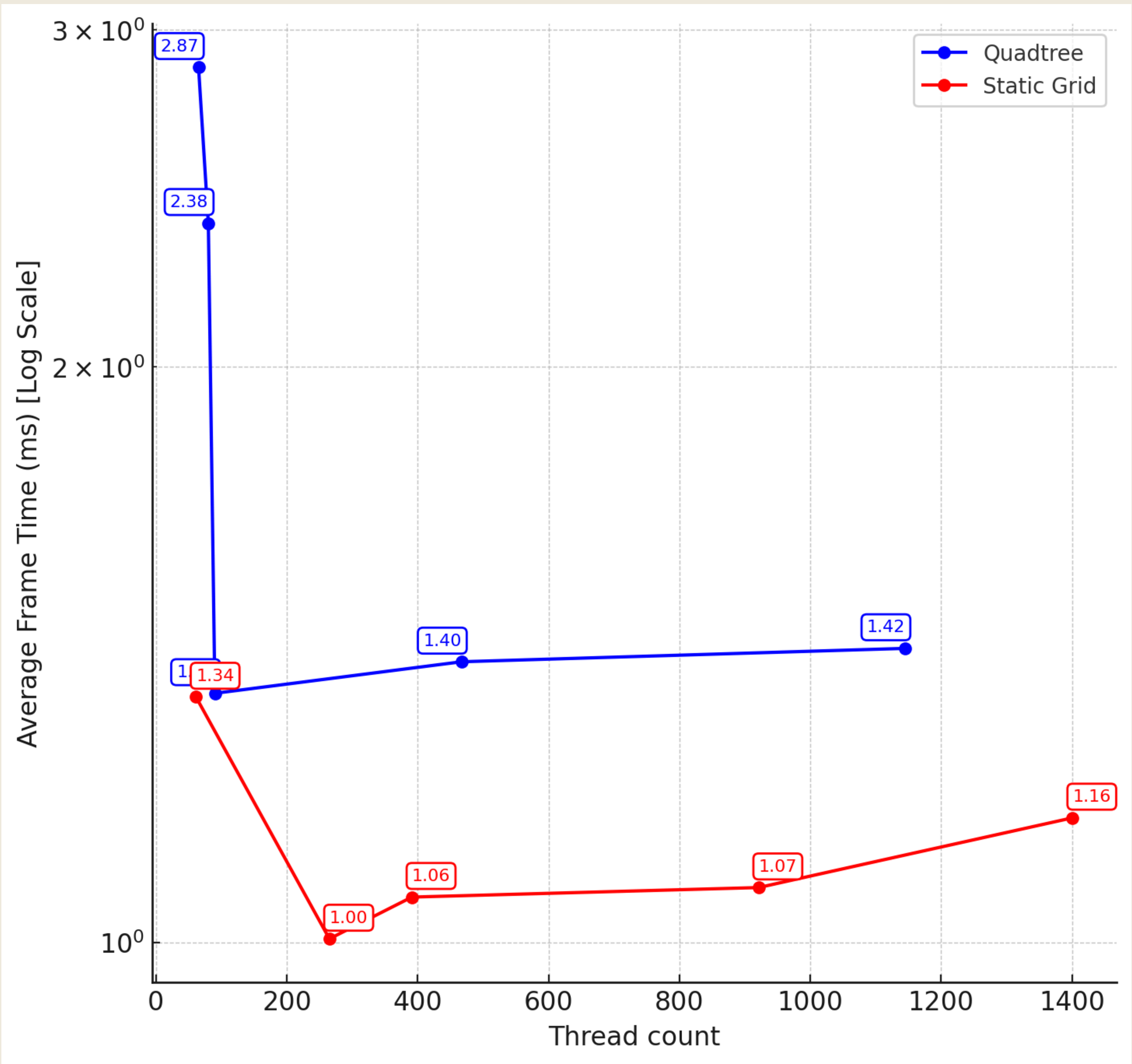


Figure 4: Imbalanced workload scene frame times

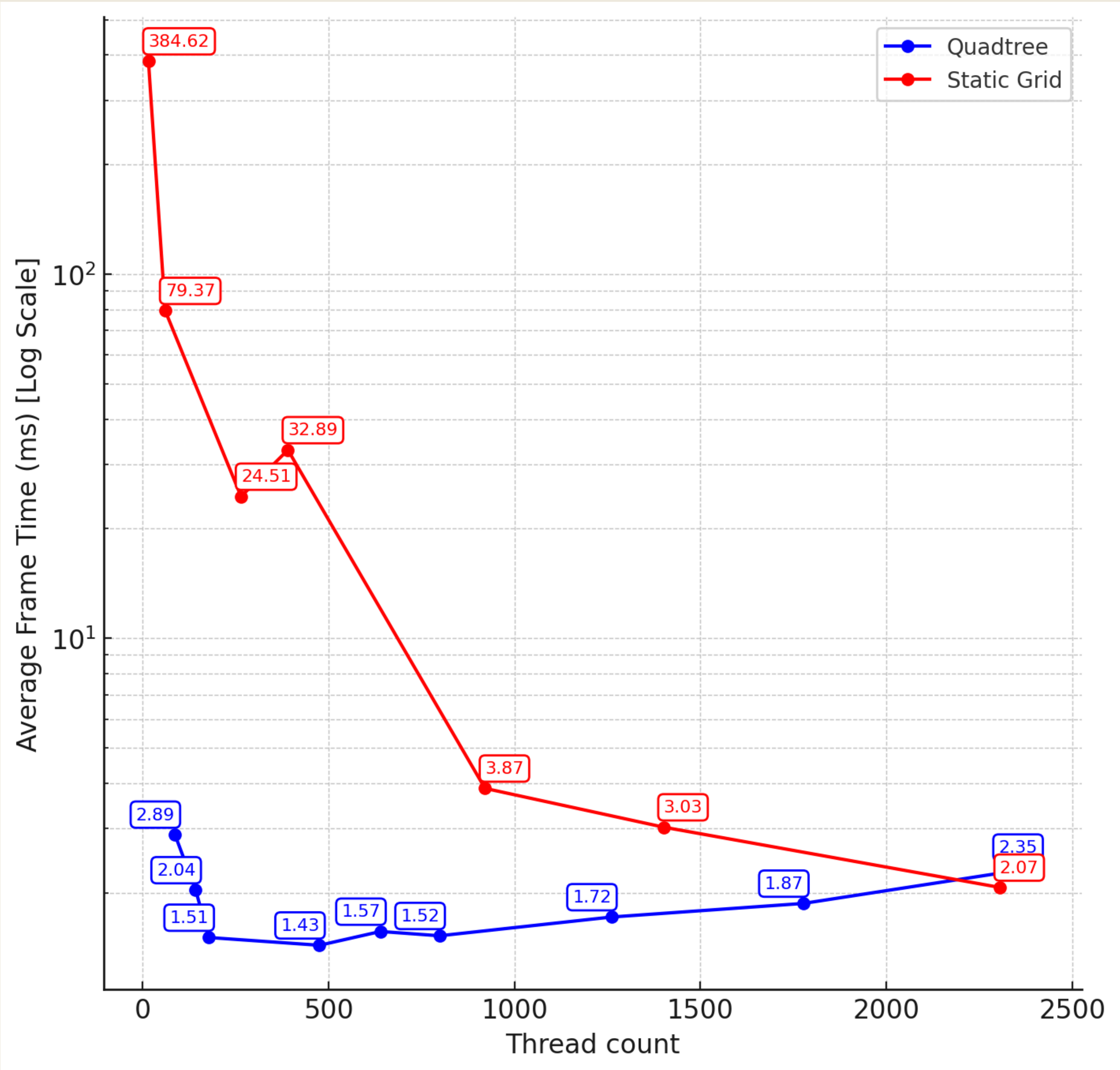


Figure 5: Imbalanced workload scene frame times

- Parallelized quadtree performs **marginally worse** for balanced distributions of workload as the quadtree grid for balanced distributions effectively resolves to resembling a static grid but requires more overhead in construction.
- Parallelized quadtree performs **significantly better** than for imbalanced distribution of workload at most thread counts due to more balanced workload.

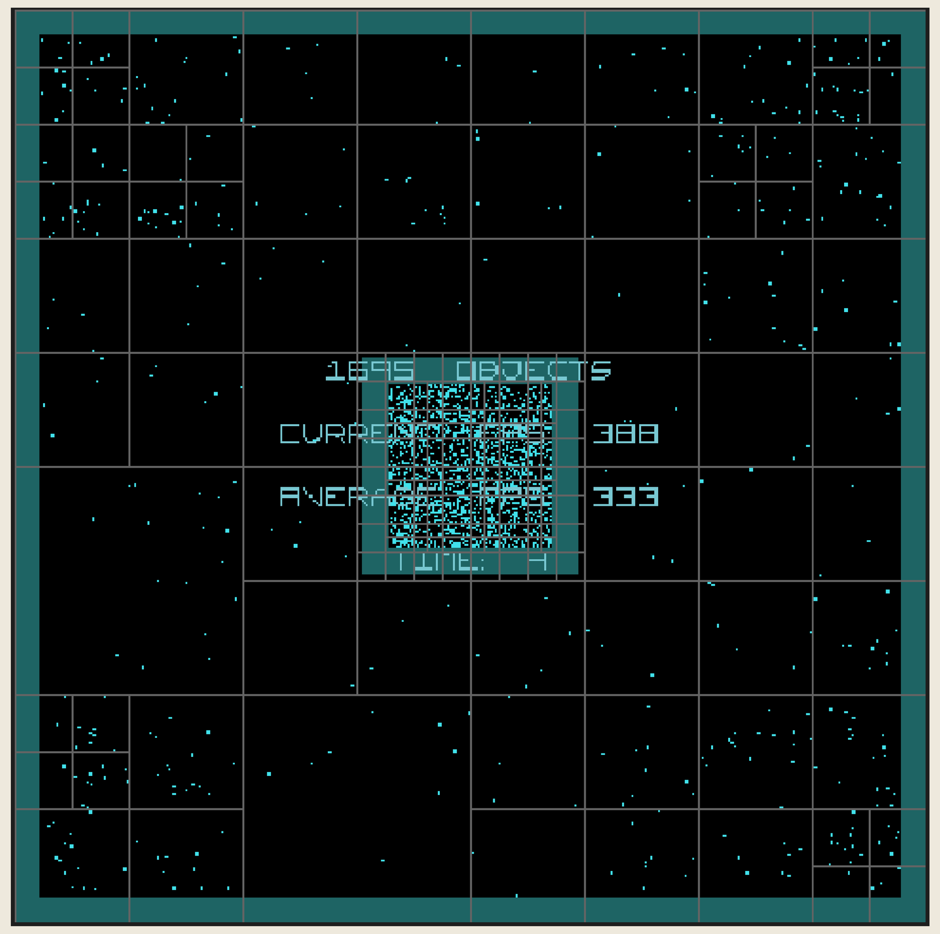


Figure 6: Imbalanced workload visualization

04. Outcomes

Code implementations

- Sequential Implementations of **Quadtree** and **Static** grid structures for entity spatial partitioning.
- CUDA Parallelized implementation of collision resolution given a list of spatial grid cells.

Key observations

- Quadtree workload partitioning performs vastly better for imbalanced entity distributions in parallelized resolution.
- CUDA parallelized resolution performs better for larger workloads.

05. Discussion & Lessons Learned

- **Parallelization Overhead**: Overhead from retrieving and copying information to CUDA kernels limits performance.
- **Specialization vs. Generalization**: More efficient parallelization would require specialized designs, sacrificing general applicability for performance.
- **Workload-Driven Structure Choice**: Quadtree excels for small, sparse, and imbalanced workloads, while static grids are ideal for large, uniform distributions.
- **Static vs. Dynamic Parallelization**: Static strategies (e.g., fixed-grid CUDA) suit predictable workloads, while dynamic approaches (e.g., Quadtree) adapt better to variability.