15-618 Final Project Milestone Report

Betweenness Centrality with CPU and GPU

Isaiah Velez Yi-Ning Huang

April 14, 2026

1 Work Completed

Yi-Ning completed the CPU side of the project by implementing the serial Brandes baseline and the OpenMP version using a shared CSR graph representation. She also built out the benchmarking and comparison workflow for the project, including the shared graph-loading semantics and the benchmark harness we now use to run the CPU baselines in a consistent way. This work established the correctness and performance baseline that the rest of the project depends on.

Isaiah completed the CUDA side of the baseline by implementing the naive GPU betweenness-centrality version with an edge-parallel mapping and atomic updates for shared vertex state, then integrating that CUDA path into the shared codebase and getting the build system working on the GHC Andrew machines. He also connected the GPU path to the same CLI and benchmark flow as the CPU implementations, added CUDA progress logging for long runs, and updated the timing pipeline so graph-building time is reported separately from BC compute time.

2 Deliverable Status

With respect to goals, the CPU and multi-core implementations are on track. However, the CUDA implementation is still behind schedule because the naive implementation only recently started working with benchmarking, and getting to that point took longer than was recently thought. With that delay, the optimization phase is not yet complete.

The nice-to-haves are still on schedule. The main issue is that, on the CUDA side, the time ratio across environment setup, debugging, benchmarking, profiling, and optimization was not estimated correctly at the start. As a result, more time than expected was spent getting the naive implementation to a stable, benchmarkable state, which pushed back the optimized CUDA work.

Please see the revised detailed schedule for the updated half-week plan for the remainder of the project.

- **April 14–16: Isaiah:** Run Nsight profiling. **Yi-Ning:** Collect serial/OpenMP benchmarks. **Shared:** Finish milestone report.
- **April 16–18: Isaiah:** Write the first conflict-aware CUDA optimization. **Yi-Ning:** Set up the final SNAP and synthetic datasets.
- **April 18–21: Isaiah:** Debug optimized CUDA and run early benchmarks. **Yi-Ning:** Finish CPU benchmark suite runs. **Shared:** Evaluate naive vs. optimized GPU scaling.

- **April 21–24: Isaiah:** Try a second conflict-aware strategy if time permits. **Yi-Ning:** Run dense/uniform graph experiments. **Shared:** Start plotting the results.
- **April 24–27: Isaiah:** Deep dive into the remaining GPU bottlenecks and draft GPU sections **Yi-Ning:** Finalize CPU charts. **Shared:** Write the implementation narrative for the report.
- **April 27–30:** Complete the final report, poster, and Autolab submission.

3 Preliminary Results

This section presents a subset of the benchmark data collected so far. Unless otherwise noted, the values below are averages over three runs, and BC compute time is reported separately from graph build time.

Benchmark Data

The first table reports average timings for the medium linear graph (`line_10000`) using `-sources 32`.

Configuration	Avg. BC Compute Time (ms)	Avg. Graph Build Time (ms)
Serial	5.663	2.678
OpenMP (1 thread)	5.641	2.637
OpenMP (2 threads)	3.015	2.556
OpenMP (4 threads)	1.723	2.653
OpenMP (8 threads)	1.256	2.689
CUDA Naive	8623.599	2.723

The second table reports average timings for the `ca-GrQc` collaboration graph, again using `-sources 32`.

Configuration	Avg. BC Compute Time (ms)	Avg. Graph Build Time (ms)
Serial	3.443	7.482
OpenMP (1 thread)	3.477	7.724
OpenMP (2 threads)	2.230	7.619
OpenMP (4 threads)	1.573	7.656
OpenMP (8 threads)	1.381	7.770
CUDA Naive	68.412	7.634

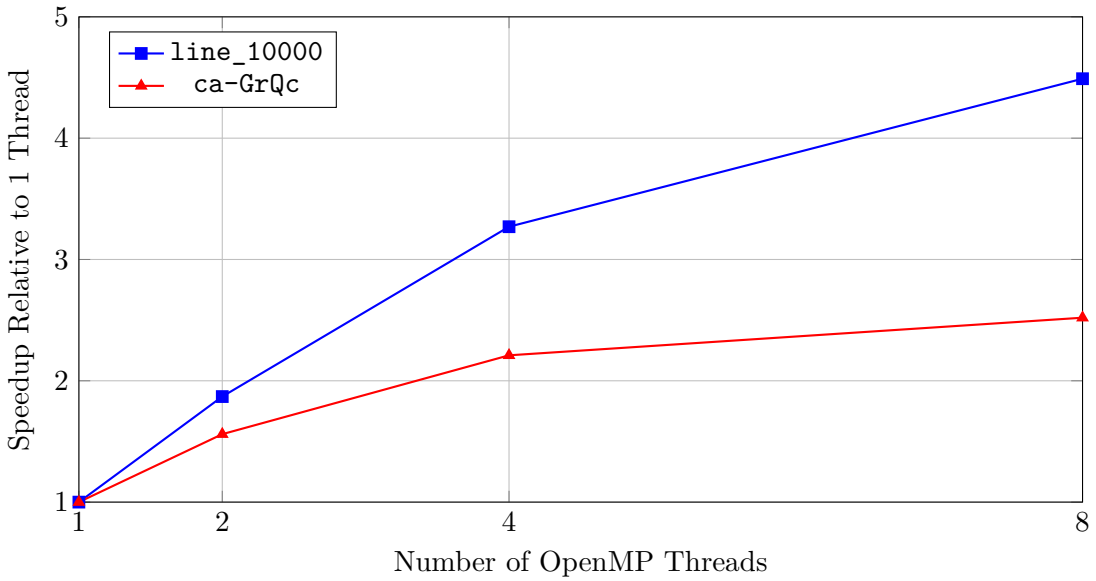

Figure 1: OpenMP Strong Scaling. The linear graph attains higher speedup due to highly uniform thread workloads, whereas the small-world graph scales worse due to thread imbalance on uneven degree distributions.

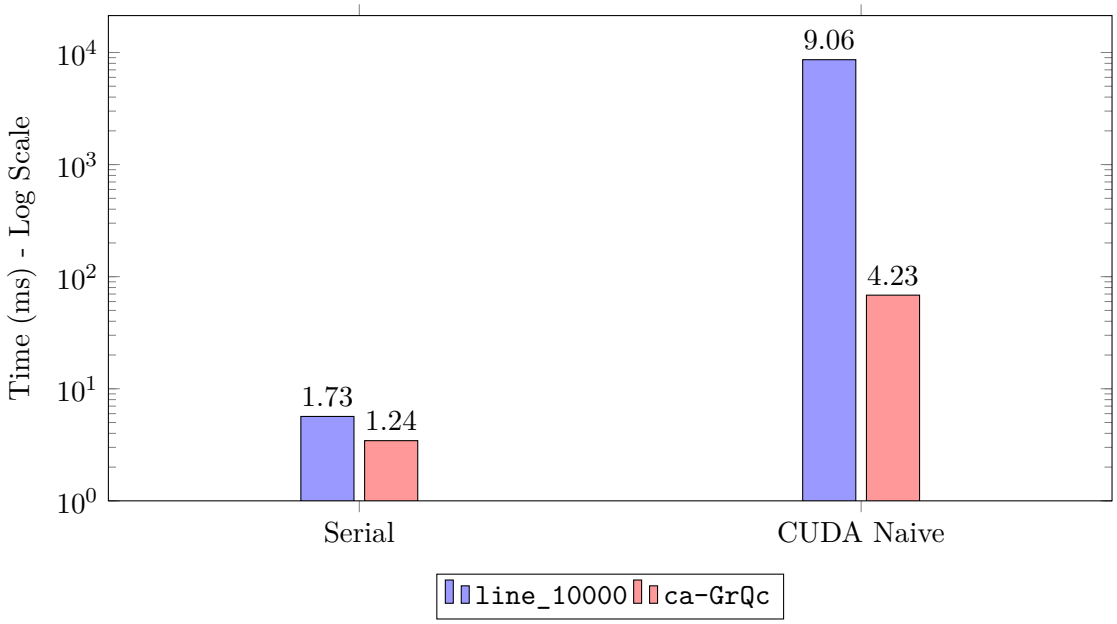

Figure 2: CPU vs GPU baseline times across topologies. The extreme slowdown on the linear graph highlights GPU kernel launch overhead for deep graphs, whereas the realistic small-world graph isolates the true atomic contention bottleneck.

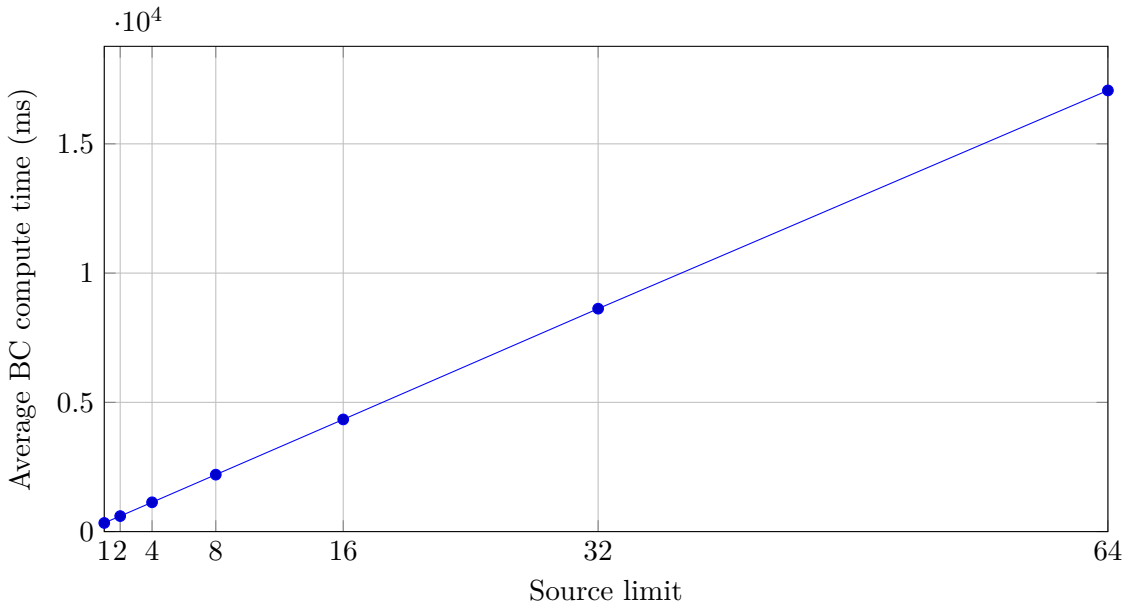

Figure 3: Average BC compute time for the naive CUDA implementation on `line_10000` scales perfectly linearly as the source cap increases, motivating parallelizing across sources in our optimization phase.

4 Updated Goals for Poster Session

For the poster session, we plan to present two concurrent stories that reflect the two main technical tracks of the project. The first story will focus on the CPU side, showing the serial baseline and the multicore OpenMP implementation, along with the benchmarking results that explain how well the CPU versions scale and where their limits appear. The second story will focus on the GPU side, following the major stepping stones from the naive CUDA implementation toward our most optimized version and explaining how each change was motivated by benchmarking and profiling observations.

The poster will therefore emphasize progression rather than just final numbers. Specifically, we plan to show:

1. **Performance Plots (CPU and GPU):** Speedup graphs comparing Serial, OpenMP, Naive GPU, and Optimized GPU across various SNAP and synthetic datasets to explain how both tracks scale.
2. **GPU Bottleneck Analysis:** Nsight profiling data showing why the naive edge-parallel CUDA version choked and evidence that our optimizations fixed these bottlenecks.
3. **Optimization Impact:** The runtime differences and intermediate measurements that motivated our optimization decisions between the naive baseline and our optimized strategies.

Our goal is that someone reading the poster can understand both how the CPU and GPU versions compare and how the GPU implementation evolved from a poorly scaling baseline into a more optimized design.

5 Concerns and Risks

At the moment, we do not have any major technical concerns about completing the project. The main issue we identified during the milestone period was improper time allocation, especially on the CUDA side where environment setup, debugging, and getting the naive implementation into a benchmarkable state took longer than originally expected.

We believe this is now manageable with the revised schedule. At this point, it is mainly a matter of allocating more time, and that is likely possible because work from our other classes is beginning to die down over the coming weeks. With the Andrew-machine build path working, the benchmark pipeline in place, and the project plan updated into half-week increments, we now have a much clearer path for the rest of the project.