15-418 Final Project Milestone Report

Linxuan Ma — linxuanm

Joseph Wan — zhilianw

1 Project Schedule

In the first half of the project, we focused on optimizing the speedup of MST in both OpenMP (optimized for arbitrary graph) and CUDA (optimized for sparse and dense graph), and establishing a variety of different graphs as benchmark for each strategy.

Most of our work are done in peer-programming; both partners share equal responsibility for each task.

Date	Task
Mar 29	Project structure + build system
April 1	Templated graph primitives with C++20 concept
April 5	Benchmark graph generator with visualization
April 8	Baseline Kruskal + OpenMP Boruvka
April 8	Batch benchmark, CSR graph variant
April 9	Batch benchmark, atomic union find
April 10	VTune benchmark, reduced contention through path-havling
April 12	Research on reducing contraction's contention
April 16	Sparse + dense CUDA implementation
April 20	Finalize all strategies + start on varying weights test
April 22	Graph decomposition based on subgraph density
April 24	Subgraph scheduling
April 28	Finish all benchmarks + final report progress check
April 30	Finish final report

For the remainder of the project, we aim to explore MST generation on graphs whose topology stays fixed while edge weights vary over time to better model dynamic, real-life workloads. This will likely be done by orchestrating our existing strategies on partitioned subgraphs.

2 Summary of Work Completed

Data: We spent the first week developing the benchmark to ensure it mirrors a dynamic, real-life workloads where graphs have varying features. We built a benchmark dataset of graph with vertices count tiny (16), small (64), medium (1024), large (65536), huge (262144), and massive (4194304). To ensure a diverse range of graph features, we generated:

- Circulant: Each vertex is connected to its k nearest neighbors in both directions on a ring, giving a regular graph with degree $2k$.
- Erdos-Renyi dense: Each possible edge exists independently with probability $p = 0.1$.
- Erdos-Renyi spase: Each possible edge exists independently with probability $p = 0.0005$.
- Grid: A $\sqrt{n} \times \sqrt{n}$ grid with 4-connectivity.
- RMat: power-law degree distribution (social network graph)

OpenMP: We used OpenMP to implement an edge-parallel Boruvka algorithm for arbitrary (i.e., unstructured) graphs. We used C++20's concept feature to quickly prototype different graph representations (adjacency list, compressed sparse rows) to measure the impact of cache locality and different union find implementations (locked, atomic with path compression, atomic with path halving) to measure the impact of contention. We reduced CPU time on atomic primitives from 45% to 30% (VTune profiling result). We are currently exploring better graph contraction methods to reduce this further in an improved strategy (current candidates are: deferring path compression, use linked-list of neighbors instead of full edge-contraction).

CUDA: We used CUDA to implement two strategies: sparse and dense. The sparse strategy is vertex-parallel, while the dense strategy is edge-parallel. The dense strategy used vectorized memory access to reduce memory overhead, packed edge data for one-shot CAS operation, and deferred path compression.

3 Goals and Deliverables

After discussing with Prof. Railing and two TAs, we changed our final goal from "low-latency graph updates" to "MST on fixed graph topology with varying weights". We will fix a graph's vertices and edge connections, then at each round, update the edges to new weights and recompute the MST. We aim to do this by doing an informed partition on the graph's structure, then use an ensemble of our OpenMP and CUDA strategies to solve each subgraph.

For the OpenMP implementation, we're able to achieve our goal of 4x speedup on 8 cores GHC on graphs with uniform degrees. This is mainly caused by contention on atomic operation, so our following goals for OpenMP will be directed towards reducing contention during contraction.

Our updated list of goals:

1. Reduce OpenMP implementation's contention; uniformly achieve 4x speedup on all basic benchmark graphs.
2. On benchmark with varying weights over time, the ensemble strategy should out-perform any single strategy in terms of speedup (to justify the orchestration overhead).
3. On dense (sub)graphs, the dense-only CUDA implementation should have over 32x speedup.

4 Plan for Poster Session

Basic Benchmark: We aim to show the speedup graphs of the OpenMP strategy on various graphs to demonstrate our progress in reducing atomic update contention regardless of graph structures. We also want to show the speedup of the CUDA Dense strategy over the number of vertices in dense graphs to show our optimizations on localizing memory accesses and reducing warp divergence in our CUDA implementation.

Varying Weights Benchmark: We would like to show a speedup graph of our ensemble to compare it against any single strategy (OpenMP or CUDA). We will also show the memory footprint of our program to compare against fully CUDA-based programs (which often uses adjacency matrices for graph representation). We will also show our benchmark graph with its vertices colored according to our graph partition to show our partitioning heuristic.

5 Preliminary Results

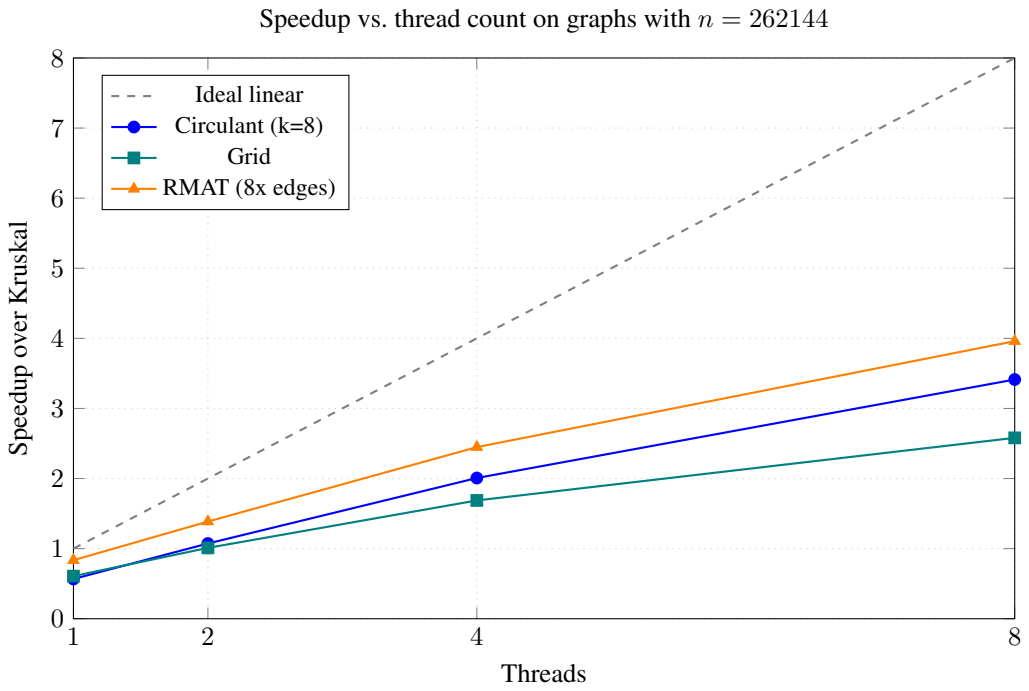

Figure 1: Speedup of OpenMP Boruvka's over sequential Kruskal's graphs with $n = 262144$

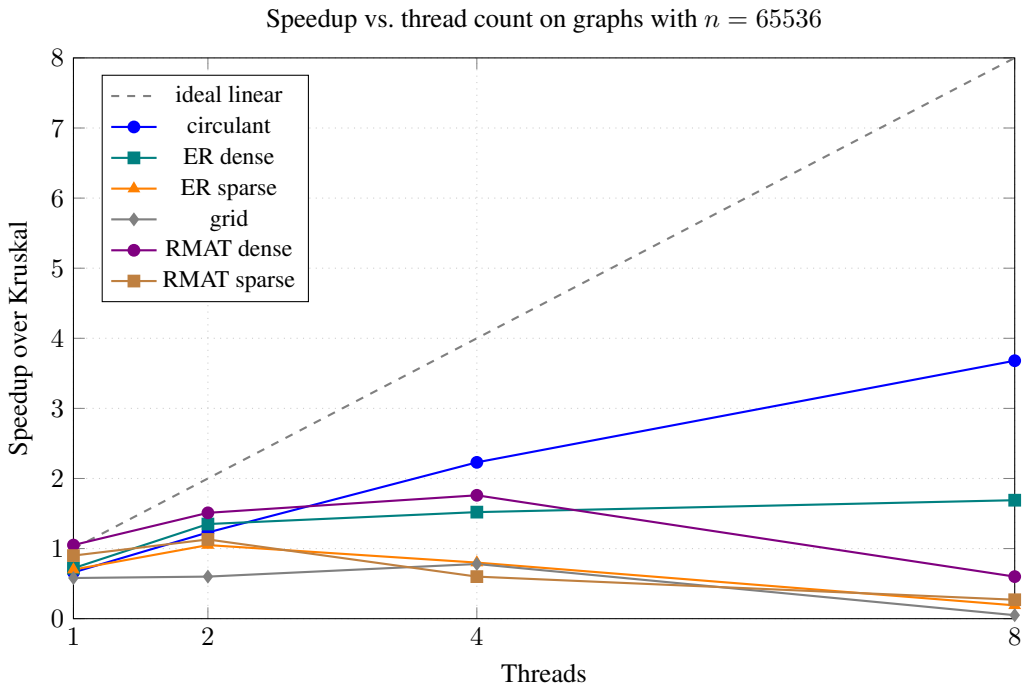

Figure 2: Speedup of OpenMP Boruvka's over sequential Kruskal's graphs with $n = 65536$

Figure 1 compares the performance of the sequential Krustal algorithm and the parallel Boruvka's algorithm using CSR for graph representation across different types of graphs that have $n = 262144$ vertices. Out of the three graph types, RMAT scales best (3.96x at 8 threads) because RMAT graphs contain hub vertices that rapidly collapses many components in a Boruvka round. Therefore, the parallel Boruvka's algorithm experiences fewer overall rounds and fewer barrier synchronizations. Circulant scales the second best (3.41x at 8 threads) because it is k-regular and gives perfect load balancing as a result, but the fact that its connectivity is local means that components grow more slowly than RMAT, therefore taking more rounds. Grid scales worst (2.58x) because each vertex has only 4 neighbors and the graph has a significant diameter, so components grow very slowly.

Figure 2 shows the speedup of the parallel algorithm on smaller graphs with $n = 65536$. The performance is unsatisfactory at 4 and 8 threads for most types of graphs because smaller graphs don't have enough parallel work to justify the initialization and thread scheduling overhead (Amdahl's Law). More importantly, there is significant contention between threads when they are currently updating the union find data structure, which is a problem that we aim to resolve.

6 Concerns

Our main concern is that our results from informed graph partitioning won't be as satisfactory. So far we've worked on non-scheduling related optimizations on general graphs, yet much of the CUDA algorithm relies on the specific graph structure (dense / sparse). While in the informed graph partition we can use k-core decomposition to segregate the graph into dense / sparse parts, we worry that the speedup will not justify how complex this is.

So far, our graph-specific results are obtained from hypothetical graphs that represents different graph features very well (e.g., degree $> 50\%$ for dense graphs). Subgraphs from partitioning will not have such well-defined features and might undermine the actual performance from what we've observed in our benchmark.

In addition, while we are using CUDA to generate structured, regular dense graphs, the MST computation is severely memory-bound, which may leads to poor speedup results.