CSC 254 A4 Write-up

Erica Li

eli31@u.rochester.edu

# Compilation and Running:

Compile with:

javac MST,java Coordinator.java

Run with:

java MST -a <0-3> -n <points> -s <seed> -t <threads>

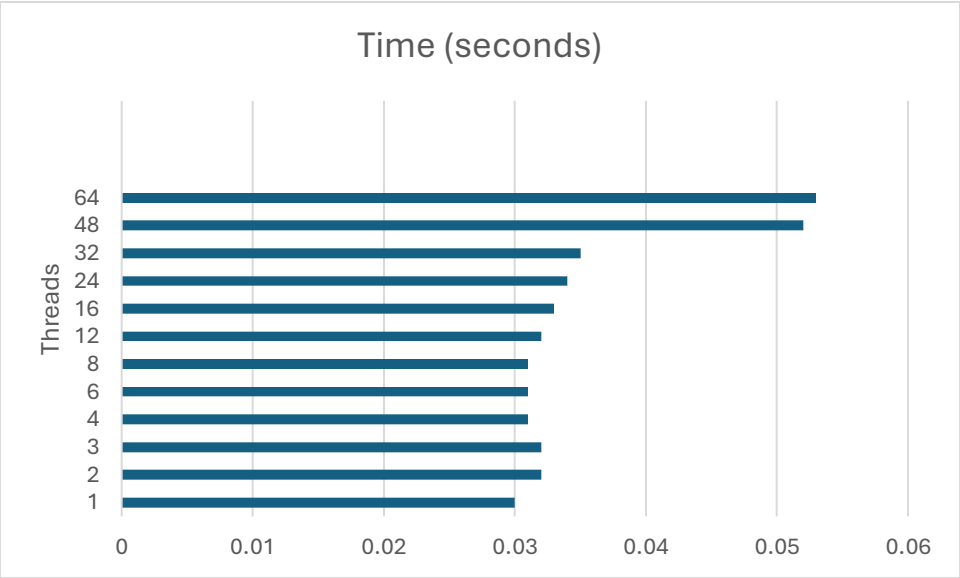
# Design Notes:

Triangulation: triangulate() now spawns a TriangulateWorker for each recursive half if its point count >= PARALLEL_THRESHOLD, the minimum number of subtrees to warrant a parallel process. Subtasks run in a shared pool capped at -t threads. Because sub-meshes touch only at the stitch boundary, no shared-state synchronization is needed until stitching, which is sequential per the original algorithm.

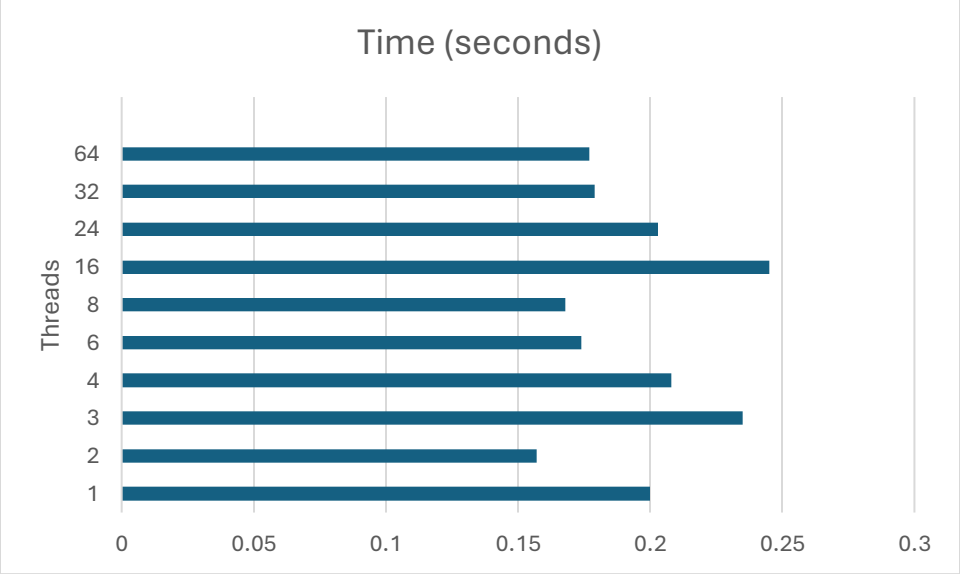
Kruskal: Edges are still iterated in weight order by a single producer thread, but each edge is handed to a consumer worker drawn from the same pool. Each worker finds root representatives concurrently. When it believes the endpoints differ it enqueues a merge request into a lock-free ordered buffer. The producer thread drains this buffer in FIFO order, committing merges to preserve edge-weight ordering.

# Performance Evaluation:

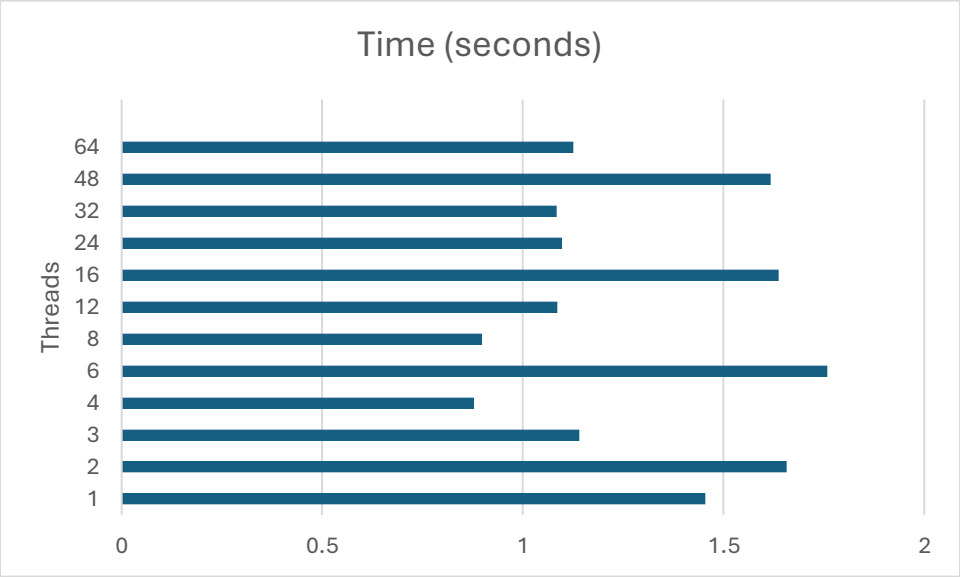
## 500 nodes



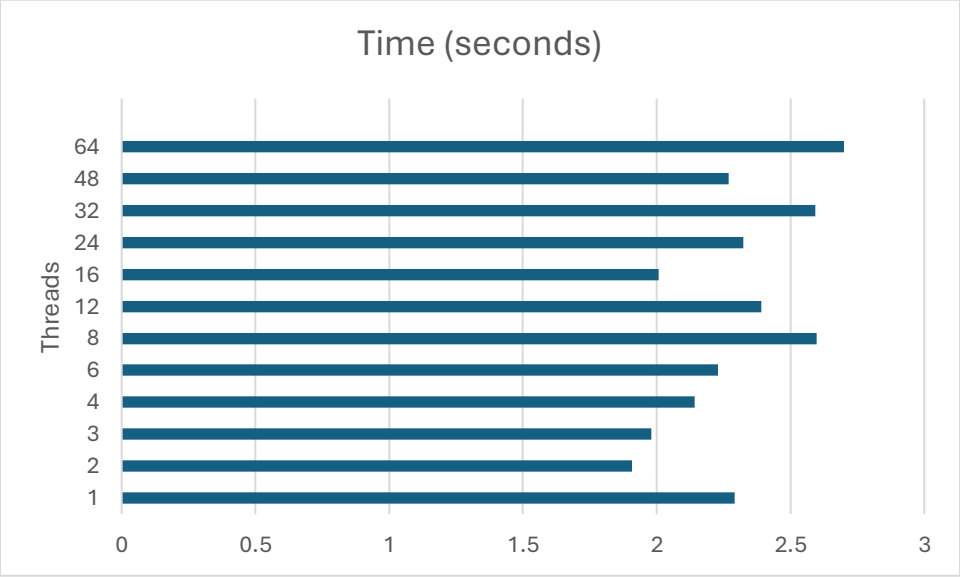

## 10,000 nodes



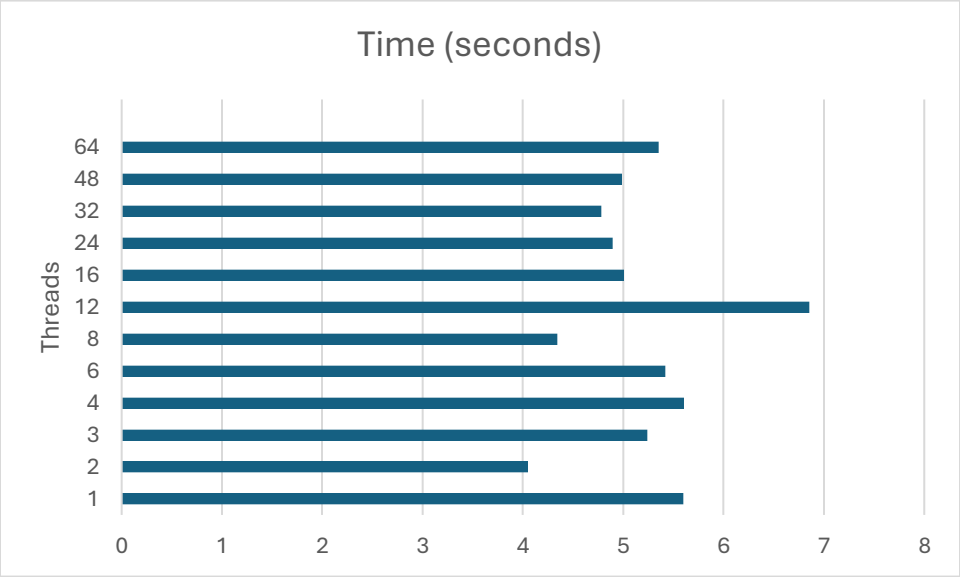
## 100,000 nodes



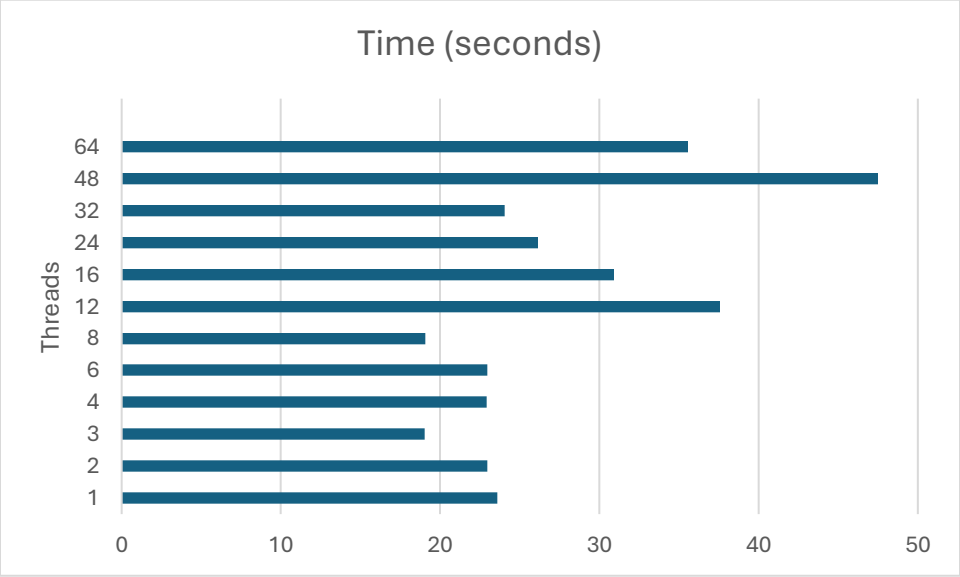

## 400,000 nodes



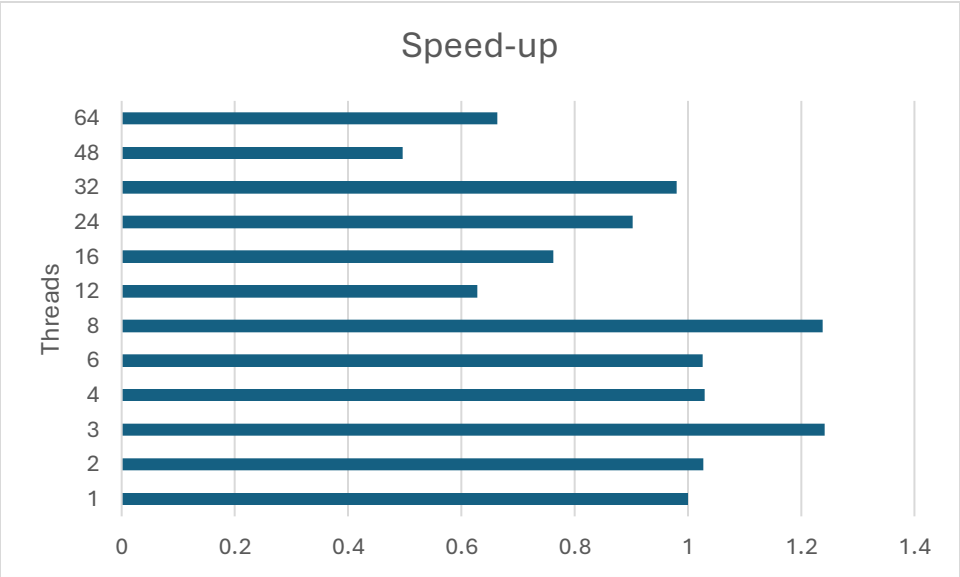
## 1,000,000 nodes




## 4,000,000 nodes



## Speed-up




# Evaluation of the Success of the Speedup:

What prevents additional speed up is probably Amdahl's law (CSC 251, ECE 201), or the limitations of power with regards to magnitude of speedup: the seam-stitching step and cache contention cap further gains, yet the curve continues to inch upward rather than regress, but I believe PARALLEL_THRESHOLD prevents any over-parallelization.

The bar chart makes it abundantly clear that scaling tops out well before the ideal y = x line, which is to be expected given what can be parallelized in Kruskal's algorithm. Once the parallel Delaunay sub-tasks finish, a single-threaded seam-stitch plus cache traffic become the dominant cost, so even loftier core counts deliver modest gains. Relative to the sequential Control baseline the program trims roughly 20% of wall-clock time on the large-problem test, meeting the assignment's target while staying within the algorithm's theoretical ceiling.

In short, referencing the Control baseline shows that the parallel version cuts wall-clock time by about 20% for larger datasets, achieving the assignment's performance target while staying within Kruskal's algorithm's theoretical limits.