Comparing Parallel Implementations of MST Algorithms

Alex Knox and Elizabeth Knox
{akknox, emknox}@andrew.cmu.edu

Introduction

Consider a connected, undirected graph $G = (V, E)$ where each edge e_i is assigned a positive weight w_i.

A **minimum spanning tree** (MST) is a connected subgraph $G' = (V', E')$ of G with $V' = V$ and $E' \subseteq E$ such that $\sum_{e_i \in E'} w_i$ is minimized over all such subgraphs.

MSTs are useful in applications including network design, clustering, and approximating the traveling salesman problem. **Because these applications involve large datasets, we are interested in scaling the performance of MST algorithms through parallelization**

Methodology

We investigate three common algorithms for finding MSTs:

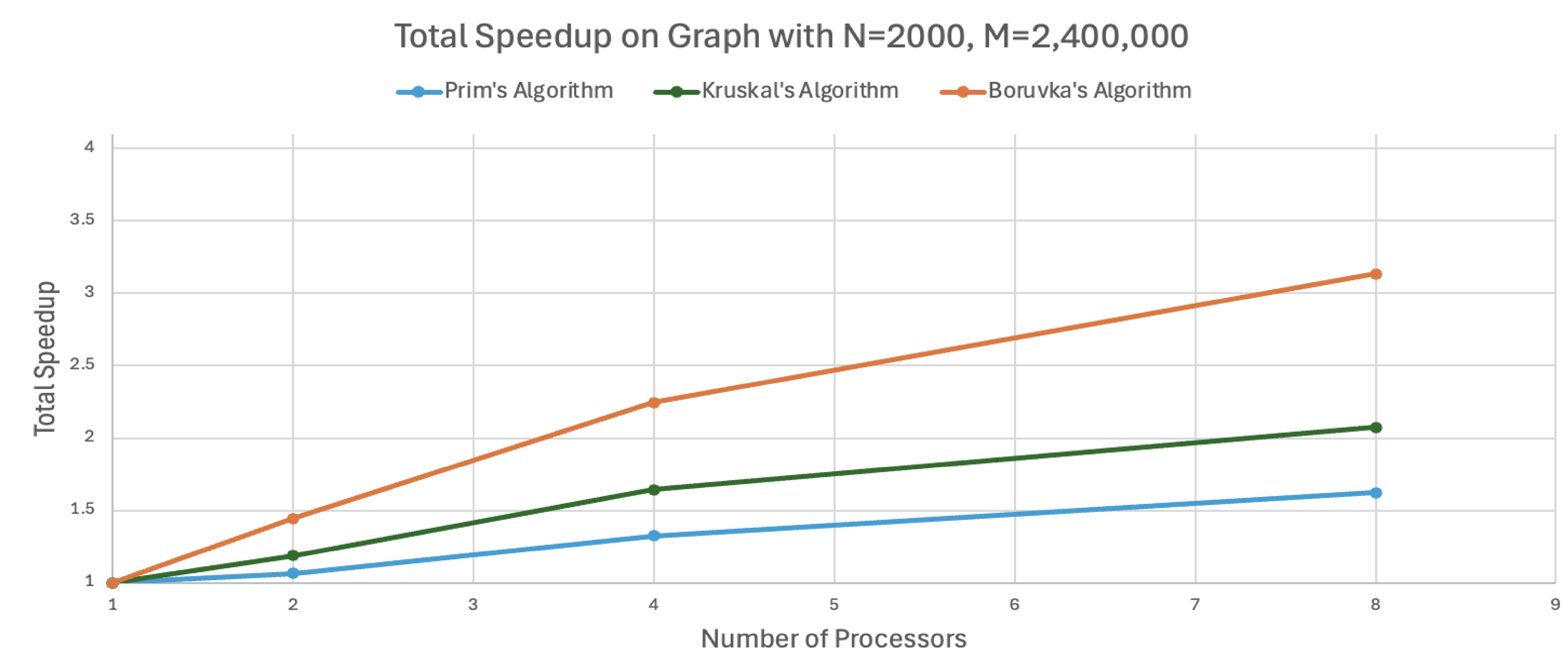
- **Prim's Algorithm**
- **Kruskal's Algorithm**
- **Boruvka's Algorithm**

Our investigation was guided by the following questions:

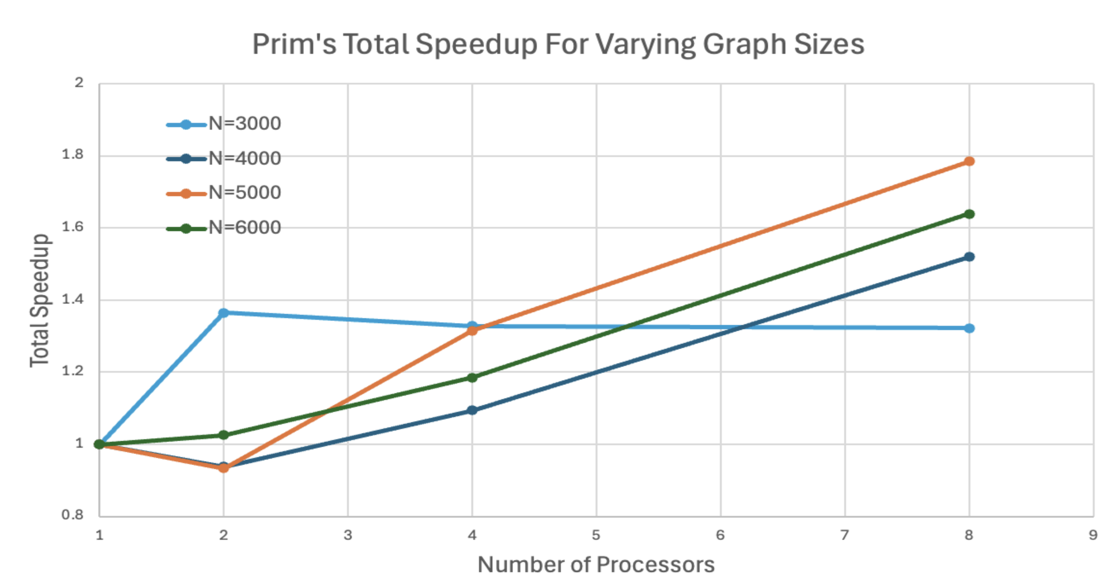
What components of the algorithm are most amenable to parallelization? What are the inherent limitations?

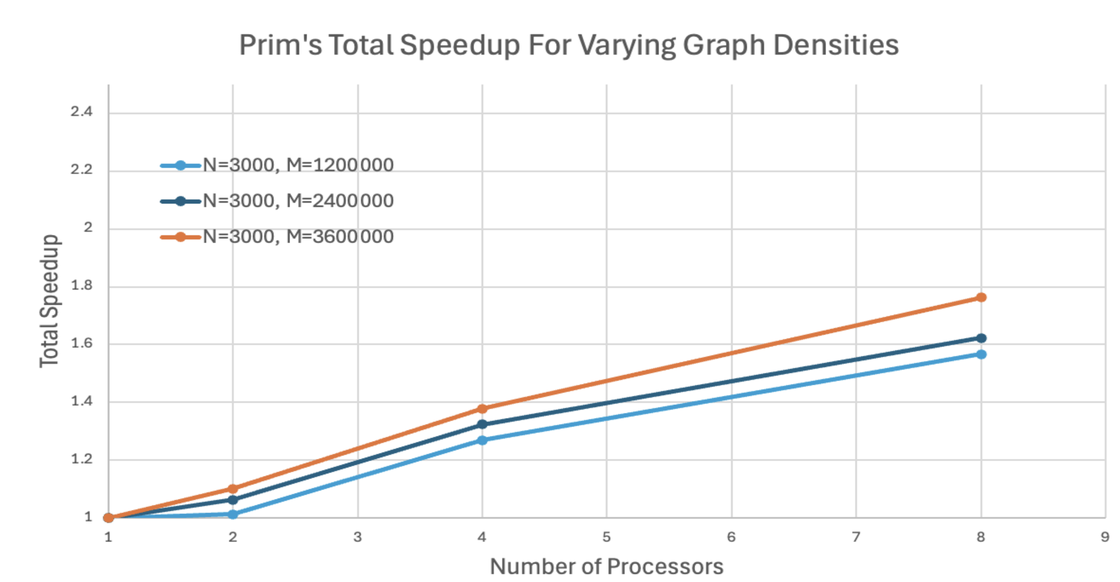
How do changes in graph parameters, such as graph size or density, affect the observed speedup of our implementations?

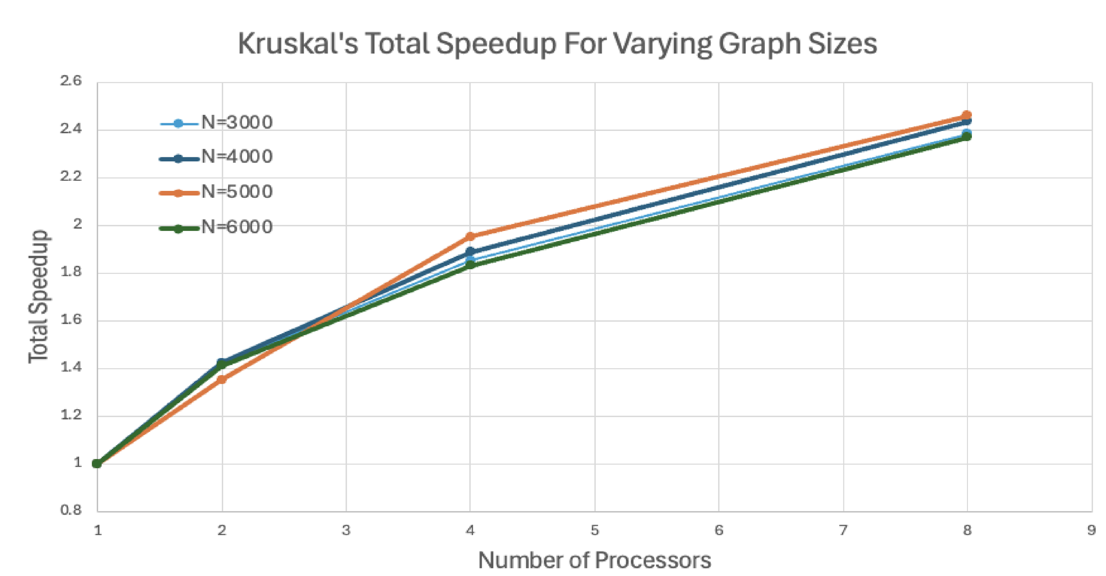
Total Speedup

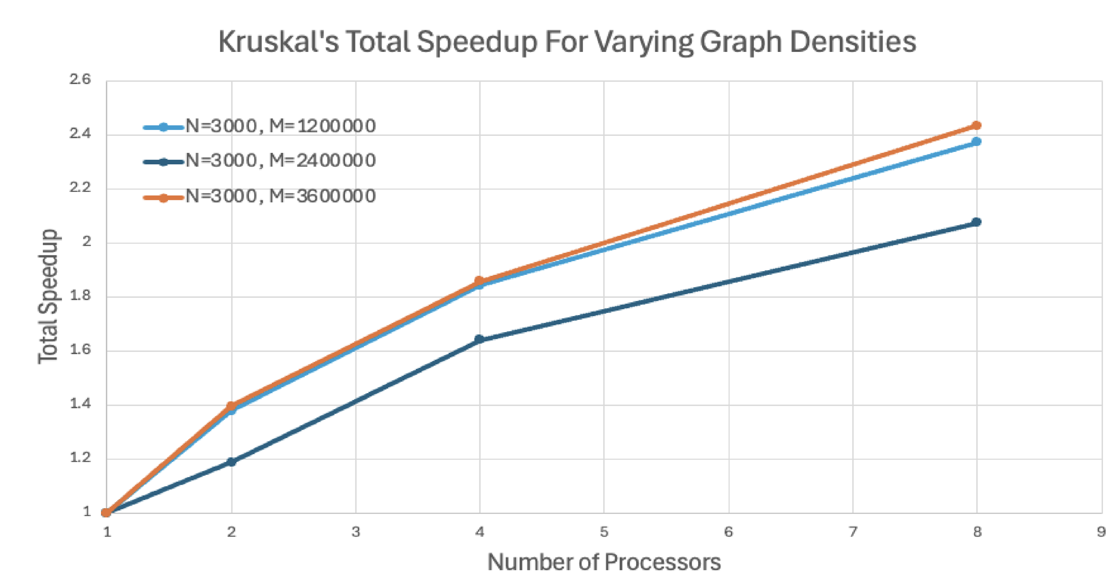


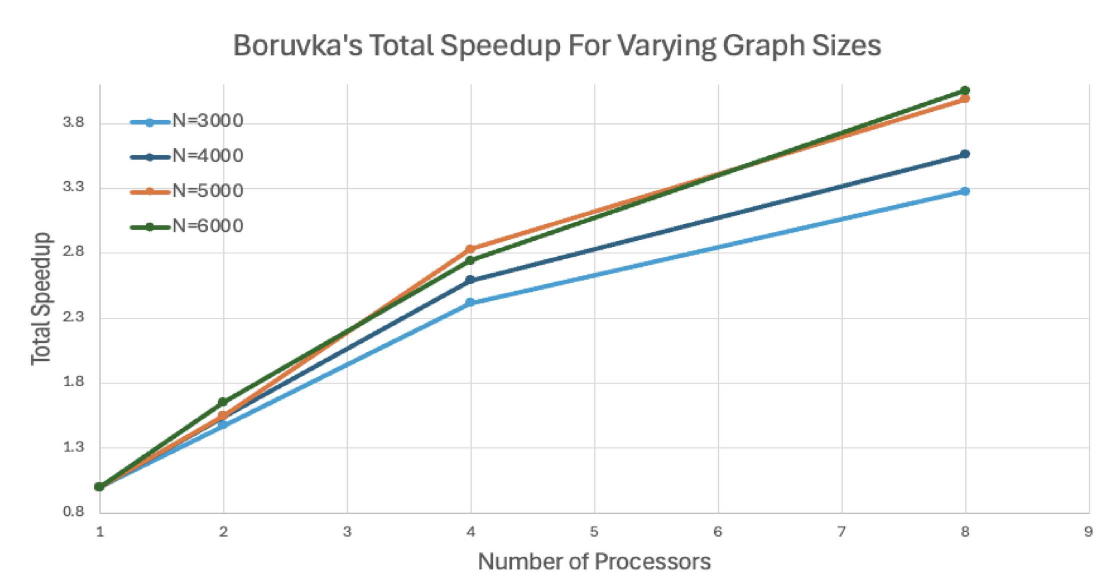
Benchmark Results

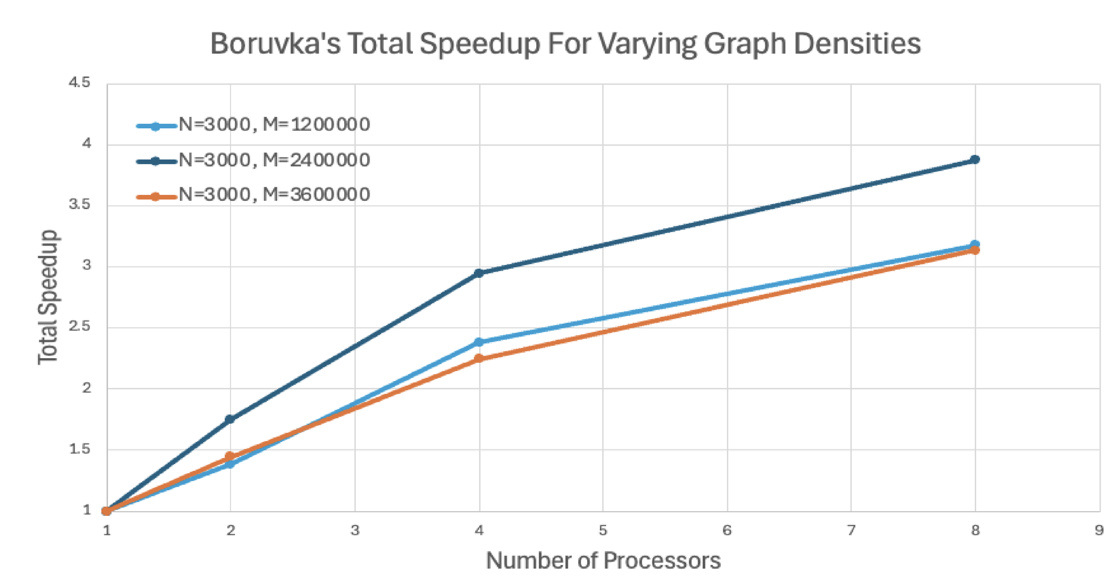












Parallel Strategies

In Prim's we use OpenMP tasks to:

- Maintain multiple priority queues for parallel insertion

In Kruskal's we:

- Sort with explicit fork/join calls.
- Eliminate expensive edges with helper threads during adjacency list traversal.

In Boruvka's we use OpenMP tasks to:

- Find the lightest outgoing edge for each vertex.
- Contract vertices and eliminate irrelevant edges to create a smaller graph.

Comparison

From the results, we find that **different sources of parallelism** in our algorithm implementations contribute to varying performance behaviors on **different graph parameters**.

Parallelism in Prim's is correlated with degree of a vertex: **scales better on dense graphs.**

Parallelism in Kruskal's is mostly independent of graph structure: **consistent speedup across graph sizes.**

Parallelism in Boruvka's helps decrease graph size: **scales better on large graphs.**